Delft Students on Software Architecture

DESOSA 2019

Arie van Deursen   Maurício Aniche   Andy Zaidman

December 2019, version 1.0

Contents

Chapter 1

Delft Students on Software Architecture: DESOSA 2019

Arie van Deursen, Maurício Aniche, and Andy Zaidman *Delft University of Technology, The Netherlands, December 20, 2019*

We are proud to present the fifth edition of *Delft Students on Software Architecture*, a collection of 25 architectural descriptions of open source software systems written by students from Delft University of Technology during a master-level course that took place in the spring of 2019.

In this course, teams of approximately 4 students could adopt an open source project of choice on GitHub. The projects selected had to be sufficiently complex and actively maintained (one or more pull requests merged per day).

During an 8-week period, the students spent one third of their time on this course, and engaged with these systems in order to understand and describe their software architecture.

Inspired by Amy Brown and Greg Wilson's Architecture of Open Source Applications, we decided to organize each description as a chapter, resulting in the present online book.

1.1 Recurring Themes

The chapters share several common themes, which are based on smaller assignments the students conducted as part of the course. These themes cover different architectural 'theories' as available on the web or in textbooks. The course used Rozanski and Woods' Software Systems Architecture, and therefore several of their architectural viewpoints and perspectives recur.

The first theme is outward looking, focusing on the use of the system. Thus, many of the chapters contain an explicit stakeholder analysis, as well as a description of the context in which the systems operate. These were based on available online documentation, as well as on an analysis of open and recently closed (GitHub) issues for these systems.

A second theme involves the development viewpoint, covering modules, layers, components, and their inter-dependencies. Furthermore, it addresses integration and testing processes used for the system under analysis.

A third recurring theme is technical debt. Large and long existing projects are commonly vulnerable to debt. The students assessed the current debt in the systems and provided proposals on resolving this debt where possible.

Besides these common themes, students were encouraged to include an analysis of additional viewpoints and perspectives, addressing, e.g., security, privacy, regulatory, evolution, or product configuration aspects of the system they studied.

1.2 First-Hand Experience

Last but not least, all students made a substantial effort to try to contribute to the actual projects. With these contributions the students had the ability to interact with the community; they often discussed with other developers and architects of the systems. This provided them insights in the architectural trade-offs made in these systems.

Student contributions included documentation changes, bug fixes, refactorings, as well as small new features.

1.3 Feedback

While we worked hard on the chapters to the best of our abilities, there might always be omissions and inaccuracies. We value your feedback on any of the material in the book. For your feedback, you can:

- Open an issue on our GitHub repository for this book.
- Offer an improvement to a chapter by posting a pull request on our GitHub repository.
- Contact @delftswa on Twitter.
- Send an email to Arie.vanDeursen at tudelft.nl.

1.4 Acknowledgments

We would like to thank:

- Anand Kanav, Arjan Langerak, and Bernd Kreynen, who did an amazing job as teaching assistants to the course
- Casper Boone and Stavrangelos Gamvrinos, who helped integrating the chapters into the DESOSA book
- Our 2019 guest speakers Mike Ciavarella, Bert Wolter, Ayushi Rastogi, Xavier Devroey, Erci Greuter, Marco di Biase, and Matthias Noback.
- Alex Nederlof and Michael de Jong who were instrumental in the earlier editions of this course.
- All open source developers who helpfully responded to the student's questions and contributions.

1.5 Previous DESOSA editions

1. Arie van Deursen, Maurício Aniche, Andy Zaidman, Liam Clark, Gijs Weterings and Romi Kharisnawan (editors). Delft Students on Software Architecture: DESOSA 2018, 2018.
2. Arie van Deursen, Maurício Aniche, Andy Zaidman, Valentine Mairet, Sander van den Oever (editors). Delft Students on Software Architecture: DESOSA 2017, 2017.
3. Arie van Deursen, Maurício Aniche, Joop Aué (editors). Delft Students on Software Architecture: DESOSA 2016, 2016.
4. Arie van Deursen and Rogier Slag (editors). Delft Students on Software Architecture: DESOSA 2015. DESOSA 2015, 2015.

1.6 Further Reading

1. Arie van Deursen, Maurício Aniche, Joop Aué, Rogier Slag, Michael de Jong, Alex Nederlof, Eric Bouwers. A Collaborative Approach to Teach Software Architecture. 48th ACM Technical Symposium on Computer Science Education (SIGCSE), 2017.
2. Arie van Deursen, Alex Nederlof, and Eric Bouwers. Teaching Software Architecture: with GitHub! avandeursen.com, December 2013.
3. Amy Brown and Greg Wilson (editors). The Architecture of Open Source Applications. Volumes 1-2, 2012.
4. Nick Rozanski and Eoin Woods. Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives. Addison-Wesley, 2012, 2nd edition.

1.7 Copyright, License and Availability

This book is available in html (easy on line reading, chapter by chapter), pdf (self contained, 726 pages) and epub (for e-reader convenience) formats.

The copyright of the chapters is with the authors of the chapters. All chapters are licensed under the Creative Commons Attribution 4.0 International License. Reuse of the material is permitted, provided adequate attribution (such as a link to chapter on the DESOSA book site) is included.

Cover image: Église Saint-Pierre de Firminy, Le Corbusier. Wikimedia

Chapter 2

The Arduino IDE

Authors: Pieter Kools, Ivo Wilms, Diwakar Babu, Arkajit Bhattacharya

2.1 Table of Contents

2.2   Introduction

Arduino is an open source electronics platform which uses simple I/O boards and a development environment to interact with the board. Arduino boards are capable of performing multiple functionalities based on the set of instructions sent to the microcontroller. The open-source Arduino software (Arduino IDE) is used to write code (C and C++) and upload it to the board. This is an analysis of the Arduino IDE architecture performed by exploring the full project on GitHub. We have aimed at providing insight into the system from different viewpoints. These viewpoints are defined and explained by Rozanski and Woods in their book *Software Systems Architecture, Working with Stakeholders using Viewpoints and Perspectives*[5]. First, a context view along with stakeholder analysis is performed, followed by development view, deployment view and finally technical debt.

2.3   Stakeholder Analysis

In this section, we identify the stakeholders involved in Arduino. The following table identifies the eleven types of stakeholders as explained by Rozanski and Woods[5.1].

Type	Stakeholder	Small Description
Acquirers	BCMI & Arduino AG Company	Massimo Banzi, CEO of the company BCMI (which acquired 100% of the Arduino AG that owns all of the Arduino trademarks), decides the future of the Arduino code[6] (of course with his board members' approvals).
Assessors	Core Developers	The Arduino company handles the legal regulations of any project that involves Arduino or using the name "Arduino" [14].
Communicators	Arduino, community	The Arduino has its own education portal (maintained by the Arduino organization) [7] providing kits (via the Arduino website) with the software (free to download from Github) and tips and knowledge on building and creating projects. People discuss Arduino projects and related code on Forums and StackExchange. YouTube is found to be a medium frequently used by many users to discuss and teach Arduino to beginners.
Developers	Core developers, integrators, community	The core developers of the Arduino software are Massimo Banzi along with Tom Igoe. The frequent and active contributors and Arduino users also play an important role in developing the Arduino code, but they do not have the permission to directly push or merge their code.
System Administrators	Community, Arduino users	Users play the role of system administrator.
Suppliers	Arduino, Github	The software packages can be directly downloaded from the official website and also from Github

Type	Stakeholder	Small Description
Maintainers	Core Developers, active contributors	The Arduino IDE keeps evolving and it is maintained by the developers and contributors by creating bug fix tasks/issues and creating/handling pull requests.
Production Engineers	Core developers	The developers of Arduino handle the production releases and run tests on new builds.
Support staff	Developers, community, Arduino organization Support team	The Arduino company has a team of support staff and communicators provide support for Arduino-based applications on various forums (for example Arduino Forums) and StackExchange.
Testers	Core Developers and community	Developers and contributors are responsible for running JUnit tests to test the code before a PR is made, merged and officially released.
Users	Hobbyists, organizations, Arduino research community groups	152 user groups with each group consisting of more than 100 members are recorded throughout the globe [8]. The applications vary from IOT to wireless applications to Robotics.

2.3.1   Beyond the classification of stakeholders by Rozanski and Woods:

The following stakeholders identified are additional stakeholders who do not match the groups in Rozanski and Woods.

- Apart from the Arduino website, local Arduino authorized *hardware suppliers* to supply the Arduino hardware.

- *End-Users* are users who use this software to work with the hardware (creating Arduino projects) which indirectly influences the Arduino's development.

- Another stakeholder that can be considered is the *people* who work at Arduino company and maintain the website which provides a space where users can share their project, based on which new features are built and added.

- Arduino blog *writers* is a space where unambiguous and reliable sources of information from users who use and help new developers to use Arduino can be found. Therefore, we can consider the bloggers as an indirect stakeholder.

- *Translators* contribute to translations for Arduino's documentation for the software on their website, who are active contributors for Arduino software.

2.3.2   Analyzing the stakeholder's involvement: Power vs Interest Grid

Mendelow's power-interest grid [13] is used to classify the groups of stakeholders that should be managed closely. Figure 1 shows that the core developers, active contributors on Github and Arduino community are stakeholders who have both high interest and power. These are the people that actively contribute to and maintain the project and need to be managed closely. Users like teachers, bloggers that use Arduino software

without many active contributions show high interest, but have very low power and must be informed well. The dependencies have high interest and minimum power. The reason being, users usually buy Arduino shields which comes with libraries and the user can easily use them without much effort. Other users like TU Delft and GSMA show very high interest in the Arduino development, but provide fewer contributions and therefore are not categorized as active contributors.
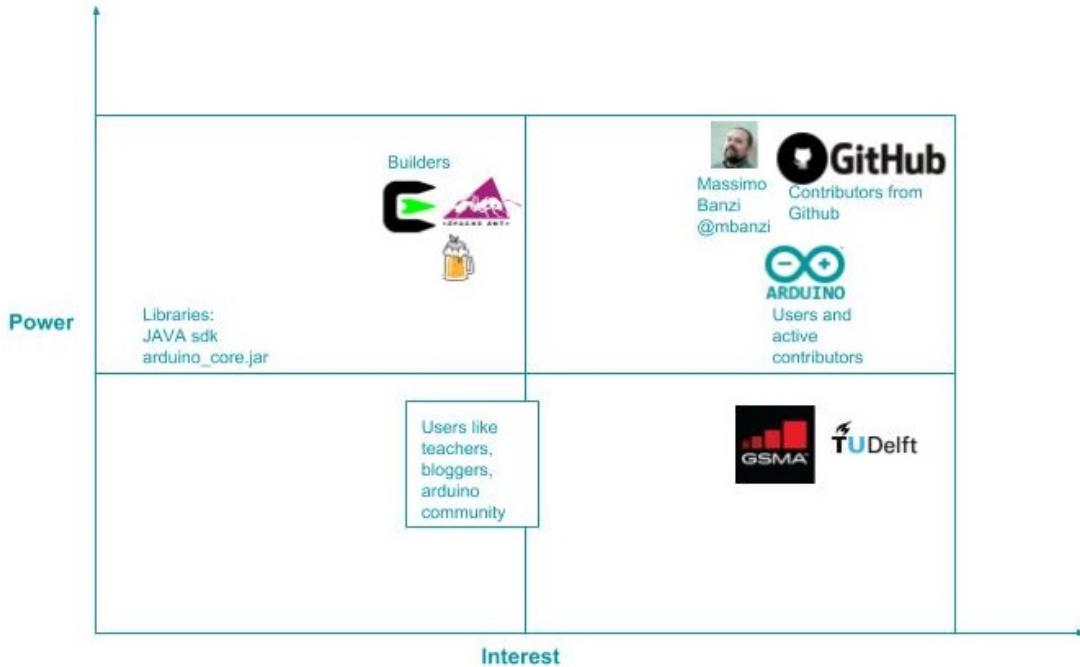
Figure 1: Power Interest Grid of the Arduino IDE where the interest of the stakeholder is shown on the horizontal axis and the power of the stakeholder is shown on the vertical axis.

2.4   Context View

The context view describes the scope and responsibilities of Arduino, i.e. its boundaries in terms of what it does or does not do. To be more precise, it defines the relationships, dependencies and interactions between the Arduino IDE and other external/internal systems, organizations, and people across these boundaries.

2.4.1   Scopes and Responsibilities

The Arduino IDE has a well-defined scope for their system. During the design process, core developers discuss what should be part of the scope for the new version. Some of the scopes for the Arduino's current versions are:

- Provide apps for MacOS, Windows and Linux.
- Arduino IDE 1.5 library format is used in tandem with its original Library Manager.

- The board manager of the Arduino IDE can be used to automatically install support for 3rd party hardware.
- New improvements are released periodically.
- Easy installation of customized libraries and command line tools.
- Sacrifice elegance of implementation over ease-of-use.
- Emphasize real use case over theoretical possibilities.
- Recognize that documentation is just as important as the code. Documentation here means the Javadocs, comments on the code that explains the particular snippets of codes.

2.4.2 Context Diagram and the External entities and Interfaces
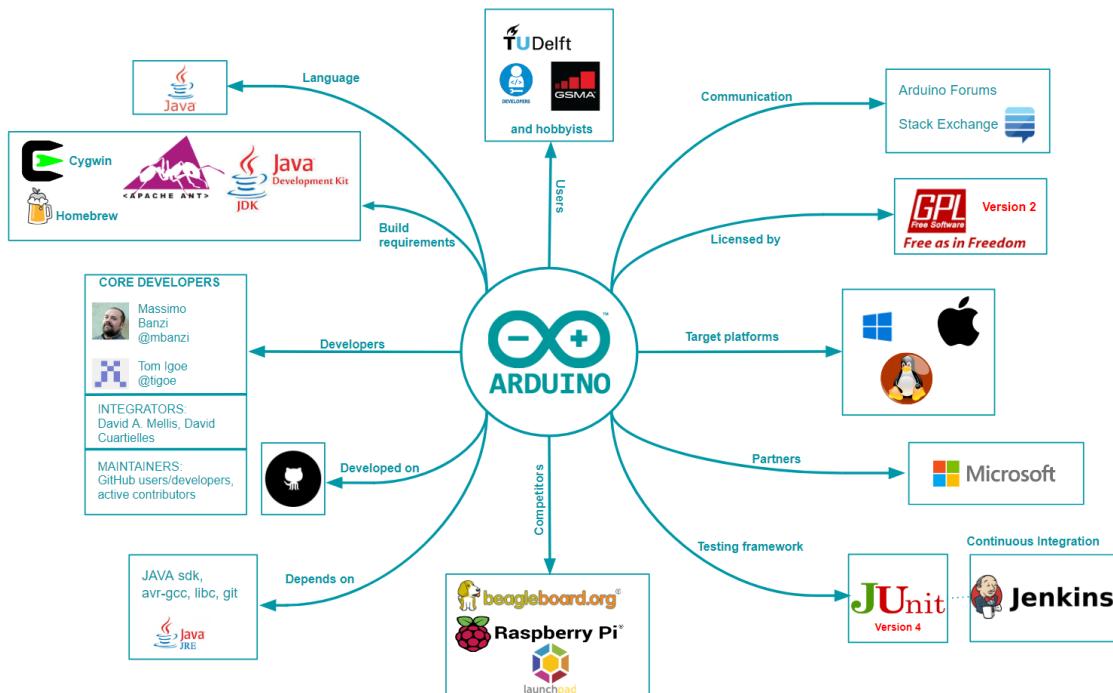
Figure 2: Context View of the Arduino IDE describing the relationships with its environment

A short explanation of some of the components in the context view model is as follows:

-**Developers**: Massimo Banzi is the main developer and co-founder of Arduino along with Tom Igoe. Additionally, active Github users/developers (for example Christian Maglie, Martino Facchin, David A. Mellis, etc.) have been a part in developing new features and maintaining the code.

-**Build Requirements**: Java Development Kit 8, Apache Ant, avr-gcc, avr-g++, git, unzip and OS-specific software such as Cygwin for Windows, Homebrew for MacOS and make and OpenJFX for Linux.[2].

-**Testing Framework**: The Arduino IDE makes use of JUnit4 to perform unit- and system tests. These tests can be executed from an IDE or by executing `ant test`.

-**Target Platforms**: The Arduino IDE was built to develop programs/apps for Arduino microprocessors on Windows, MacOS and Linux platforms.

-**Communicators**: Arduino users (teachers, bloggers and developers) communicate mostly on the dedicated
Arduino forums and on Stack Exchange.

-**Dependencies**: The Arduino IDE has a lot of dependencies as seen here and here (.jar files).

-**Language**: The Arduino IDE is written in Java and is designed for C/C++ development for Arduino
microprocessors.

2.5   Development View
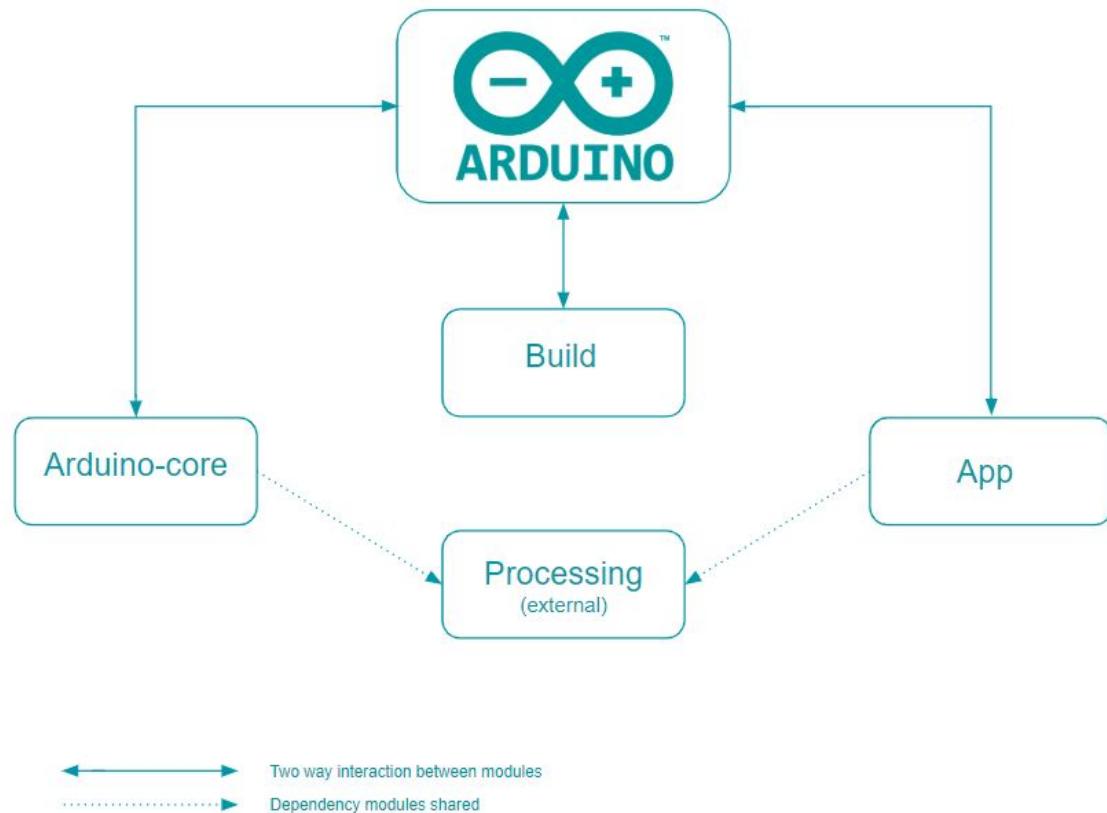
Figure 3: High-level architecture of the Arduino IDE.

Figure 3 shows the high-level architecture of the Arduino IDE, consisting of main modules `arduino-core`,
`app` and `build`. The functionality of these modules will be explained in the Module organization section,
followed by a look into the Common Design Model. The Codeline model then takes a deeper look into the
project structure and last, the Stakeholders concerned with Development View are identified.

2.5.1 Module Organization
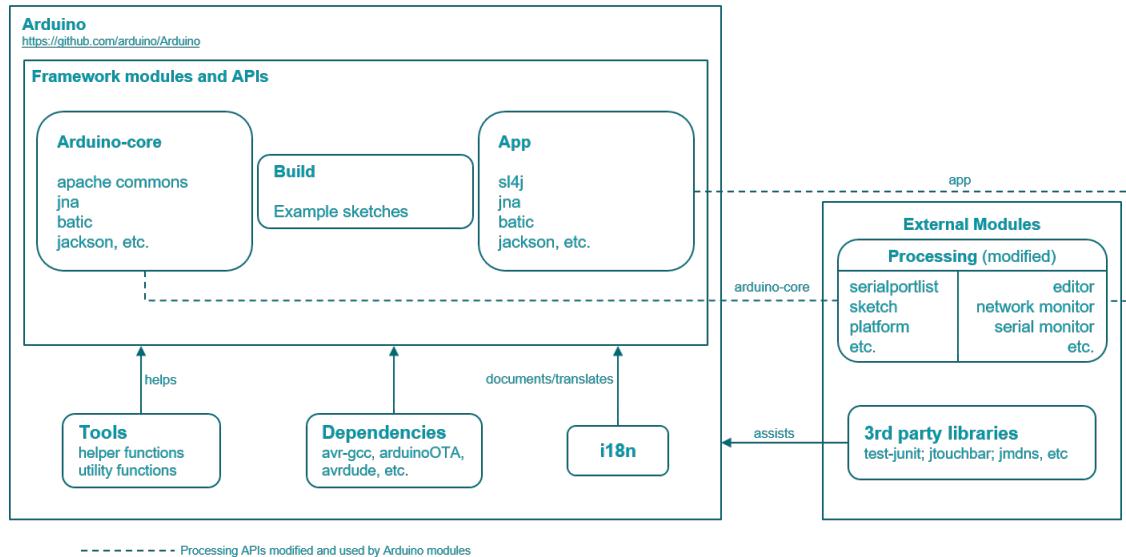
Figure 4: Module Organization of the Arduino IDE.

Figure 4 shows the various modules of the Arduino IDE and how they relate to each other.

Modules - *arduino-core (Java)*
This module contains the code required for the core IDE features. It does not contain a GUI and can be executed from the command line. The most important features are providing a serial connection to a through USB connected Arduino microprocessor, merging C code sketches with used libraries, combining compiled sketches with the right bootloader for the used Arduino microprocessor type and providing a translation framework for any feedback shown to the end user.

- *app (Java)*
  This module is built on top of the `arduino-core` module and provides the GUI that we know as the Arduino IDE. The most important components of this GUI are the code editor, the serial monitor, the serial plotter and the library manager.

- *build (binaries/images/text)*
  This module contains the additional non-Java files that are used in combination with `arduino-core` and `app` to build the distributable zip of the Arduino IDE as it can be downloaded by end users. The most important files are:

  – *The Apache Ant build file*: This file contains the build configuration of the entire project.
  – *Example sketch references*: This is a list of example sketch repositories for programming Arduino microprocessors and `sha` hashes to validate that these automatically cloned repositories indeed contain the expected content (version).
  – *OS-specific drivers*: The OS-specific drivers that allow connecting to Arduino microprocessors through USB.
  – *Native libraries*: Libraries that are required by either `arduino-core` or `app`, such as the GCC C compiler and `.dll` files on Windows for getting certain default directories.
  – *Art*: Images used by `app`.

Dependencies

The project is written in Java and has the following build dependencies: - *Apache Ant:* Used for building the project. - *Java Development Kit (JDK):* Used for compiling Java sources. - *Arduino C libraries:* These are libraries that can be used by end users to write code for Arduino microprocessors. These are: Ethernet, GMN, Stepper, TFT, WiFi, arduino, Bridge, Robot_Control, Robot_Motor, RobotIRremote, SpacebrewYun, Temboo, Esplora, Mouse, Keyboard, SD, Servo, LiquidCrystal and Adafruit_CircuitPlayground. In essence, all libraries except the `arduino` library can be omitted. The `arduino` library contains functions that allow the user to access Arduino microcontroller functionality without having to perform direct register manipulation. - *GCC compiler:* Users program Arduino microcontrollers in C. The GCC compiler is used to compile that C code to machine code that runs on different Arduino microprocessor architectures. - *launch4j:* Used to wrap the Java project into a Windows executable. - *Open source Java libraries:* Several more generic open source Java libraries are used. The jars with their licenses can be viewed in the repository at `app/lib` and `arduino-core/lib`.

External dependencies

The `arduino-core` and `app` modules both contain a `processing.app` package containing partially modified classes from the Processing Development Environment (PDE). Every now and then when the PDE project updates, those updates are also manually merged into the copied classes. In an attempt to keep this merging process as simple as possible, Arduino IDE developers attempt to keep the signature of these classes as similar to the original as possible. Modifications to these classes are made to fulfill the need of Arduino IDE specific features.

2.5.2   Common Design Model

Every version of Arduino is found to be similar due to the commonality factor, which is provided by the constraints of the development of the Arduino IDE. The main reason for this is to reduce risk and duplication of effort, in combination with increasing the system's overall coherence.

Common Processing

The modules `arduino-core` and `app` consist of unique packages, including adapted packages from the Processing Development Environment (PDE), which were modified to fit the Arduino IDE's needs. No processes are shared.

Internationalization

The Arduino community has members from different parts of the world. The i18n translator is used to make the Arduino software more accessible to people who speak different languages.

Standardization of design

The Development policy [15] is well documented in each area of open-source project development such as issue creation, making pull requests and code optimization. This helps developers and contributors to maintain the design standards throughout the projects life. Some design standards of Arduino are as follows:

- *Issues:* Issues are used for reporting bugs and proposing/discussing new features. The issue tracker is used to keep track of these bugs/features and allow everyone to view the discussion and code progress. When the issues are relatively easy, it normally takes 3 - 5 days to be resolved by developers. However, many issues stay open for a very long time. Issue 134 is the oldest open issue. It has been open 6.5 years. Any bugs found in the forums are also found by the developers and contributors and added to issue tracker and linked to the respective forum page.

- *Submitting Patches:* Optimizations and bug fixes are maintained separately (sometimes they are considered a low priority). Since fixing some bugs can break the flow and functionality, Arduino expects the changes to be tested on processors/boards before adding to the code. In the Arduino, issues related to code optimization are found to be resolved sooner than the bug report issues.

Customisation of libraries: Certain important libraries are maintained by the developers' team since they are an essential part of the code. These libraries are contributed by the community and can be seen in the list. Any customized libraries can be sent to the Arduino developer forum for any suggestions or possible inclusion as a new entry.

- *Pull requests:* Pull requests facilitate the contribution of additional features, modification of existing features and fixing of bugs in the Arduino IDE. The points that influence the pull request decision-making process in the Arduino IDE are as follows:

1. *Desirability* - When added features would not seem to improve the Arduino software, questions about this were asked and the PR authors (and other people) were given the opportunity to explain why the PR is an improvement for the Arduino software. Even though some features in the analyzed PRs were questioned, none of these PRs were rejected for this reason.
2. *Code quality* - Developers often discussed the implementation of features. This was often about design choices such as: add or do not add an abstraction layer, where to add new API functionality, implement differently to provide more useful errors to end users and add some feature/bugfix in the targeted module or rather fix it in a dependency of that module.
3. *Test Results* - The PR authors nearly always update their PRs or supply additional information when a failed user/build test is reported. Though not directly discussed, it is likely that PRs with known errors are less likely to be merged. Exceptions to this hypothesis were OS/setup-specific errors that were not severe for the end user (minor graphical bugs, the installer not generating a desktop shortcut on Linux, etc).

2.5.3 Codeline model

The codeline models in terms of the source code structure, release process, configuration management, build and testing processes are discussed below.

⊿ 🗂 arduino-core [Arduino ant-build-bugfix]
  ⊿ 📁 src
    ⊿ 🎛 cc.arduino
      ▷ 🎛 contributions
      ▷ 🎛 files
      ▷ 🎛 filters
      ▷ 🎛 i18n
      ▷ 🎛 net
      ▷ 🎛 os.windows
      ▷ 🎛 packages
      ▷ 🎛 utils
      ▷ 🗋 Compiler.java
      ▷ 🗋 CompilerProgressListener.java
      ▷ 🗋 CompilerUtils.java
      ▷ 🗋 Constants.java
      ▷ 🗋 DefaultUncaughtExceptionHandler.java
      ▷ 🗋 LoadVIDPIDSpecificPreferences.java
      ▷ 🗋 MessageConsumerOutputStream.java
      ▷ 🗋 MyStreamPumper.java
      ▷ 🗋 ProgressAwareMessageConsumer.java
      ▷ 🗋 UploaderUtils.java
    ⊿ 🎛 edazdarevic.commons.net
      ▷ 🗋 CIDRUtils.java
    ⊿ 🎛 processing.app
      ▷ 🎛 debug
      ▷ 🎛 helpers
      ▷ 🎛 i18n
      ▷ 🎛 legacy
      ▷ 🎛 linux
      ▷ 🎛 macosx
      ▷ 🎛 packages
      ▷ 🎛 tools
      ▷ 🎛 windows
      ▷ 🗋 BaseNoGui.java
      ▷ 🗋 I18n.java
      ▷ 🗋 Platform.java
      ▷ 🗋 PreferencesData.java
      ▷ 🗋 Serial.java
      ▷ 🗋 SerialException.java
      ▷ 🗋 SerialNotFoundException.java
      ▷ 🗋 SerialPortList.java
      ▷ 🗋 Sketch.java
      ▷ 🗋 SketchFile.java

⊿ 🗂 > app [Arduino ant-build-bugfix]
  ⊿ 📁 > src
    ⊿ 🎛 cc.arduino
      ▷ 🎛 contributions
      ▷ 🎛 packages
      ▷ 🎛 view
      ▷ 🗋 ConsoleOutputStream.java
      ▷ 🗋 UpdatableBoardsLibsFakeURLsHandler.java
    ⊿ 🎛 > processing.app
      ▷ 🎛 forms
      ▷ 🎛 helpers
      ▷ 🎛 javax.swing.filechooser
      ▷ 🎛 macosx
      ▷ 🎛 syntax
      ▷ 🎛 tools
      ▷ 🗋 AbstractMonitor.java
      ▷ 🗋 > AbstractTextMonitor.java
      ▷ 🗋 Base.java
      ▷ 🗋 Editor.java
      ▷ 🗋 EditorConsole.java
      ▷ 🗋 EditorHeader.java
      ▷ 🗋 EditorLineStatus.java
      ▷ 🗋 EditorStatus.java
      ▷ 🗋 EditorTab.java
      ▷ 🗋 EditorToolbar.java
      ▷ 🗋 NetworkMonitor.java
      ▷ 🗋 NewBoardListener.java
      ▷ 🗋 Preferences.java
      ▷ 🗋 PresentMode.java
      ▷ 🗋 Resources.java
      ▷ 🗋 RunnerListener.java
      ▷ 🗋 SerialMonitor.java
      ▷ 🗋 SerialPlotter.java
      ▷ 🗋 SketchController.java
      ▷ 🗋 TextAreaFIFO.java
      ▷ 🗋 Theme.java
      ▷ 🗋 UpdateCheck.java

Figure 5: Source code structure of the app and arduino-core module source directories.

▲ 🗂 > app/test
  ▲ 🔲 cc.arduino
    ▷ 🔲 contributions
    ▷ 🔲 i18n
    ▷ 🔲 net
    ▷ 🔲 packages
  ▲ 🔲 > processing.app
    ▷ 🔲 debug
    ▷ 🔲 helpers
    ▷ 🔲 linux
    ▷ 🔲 macosx
    ▷ 🔲 syntax
    ▷ 🔲 tools
    ▷ 🔲 windows
    ▷ 📄 AbstractGUITest.java
    ▷ 📄 AbstractWithPreferencesTest.java
    ▷ 📄 AutoformatProducesOneUndoActionTest.java
    ▷ 📄 AutoformatSavesCaretPositionTest.java
    ▷ 📄 AutoformatTest.java
    ▷ 📄 BlockCommentGeneratesOneUndoActionTest.java
    ▷ 📄 CommandLineTest.java
    ▷ 📄 DefaultTargetTest.java
    ▷ 📄 HittingEscapeOnCloseConfirmationDialogTest.java
    ▷ 📄 ReduceIndentWith1CharOnLastLineTest.java
    ▷ 📄 ReplacingTextGeneratesTwoUndoActionsTest.java
    ▷ 📄 TestHelper.java
    ▷ 📄 > UpdateTextAreaActionTest.java
  📄 Keypad_mac.zip
  📄 Keypad_with_hidden_files.zip
  📄 optiboot_atmega328.hex
  📄 sketch.hex
  📄 Test.zip
  📄 Test2.zip

Figure 6: Source code

structure of the app module test source directory.

The source code structure in figures 5 and 6 show the main packages and top-level classes used to build and test the Arduino IDE.

Building and Testing
The Arduino IDE is built using Apache Ant in combination with a Java Development Kit (JDK) and OS-specific tools for generating the executable. Building happens through executing `ant dist` (build and generate distribution zip), `ant run` (build and start) or `ant build` (build only) from the command line in the `build` directory. This effectively executes the Apache Ant build task as defined in `build/build.xml`. This build task also takes care of dependency management tasks such as downloading dependencies, unpacking dependencies, cloning git repositories and checking SHA hashes of downloaded dependencies (the .sha files are supplied in the repository). The Jenkins Continuous Integration is used to generate hourly builds and beta builds.

The `arduino-core` module does not contain any tests and the `app` module contains JUnit4 tests. The `ant test` command runs these tests. The tests are not very robust: while we tested it the results ranged from 4 to 17 test failures depending on the operating system and whether were spaces in the path. Oddly enough, these tests are not automatically executed when building locally or through the Jenkins CI.

Release Process
Production engineers are majorly responsible for monitoring the releases and also control them on Github. The software release process is not much different from any other projects on Github. 1. Changes are made on separate branches in a fork of the repository. 2. Merge request is created by a contributor and eventually checked by the production engineers. 3. Version number is used to keep track of releases. 4. Draft release is created with release notes. 5. Rebase the branch onto master and merge the pull request. When the pull request needs rebasing and the developer who performs the merge does not have write access to the pull request, it is closed and then rebased onto master on another branch and merged from there.

After these steps, the production engineer ensures that the release notes are updated properly. The release takes place after a set of pull requests has been merged.

The Arduino IDE is managed by a small set of developers and contributed to by a larger group of users and developers. Version control is used to document additional features and non-backward compatible changes. Pull requests for such changes are postponed until all changes combined are worth a version bump. Since pull requests are always built and often user-tested before merging to the master branch, the technical integrity of the master branch is maintained (it is nearly always stable). While developers do often discuss features, they less often discuss how they should be implemented. This could be because contributors are doing a good job, but it could also mean that developers mainly care about the final result. The Arduino IDE project does not have any static analysis tools and does not come with a code style template. While most of the code is consistent, there are still quite some inconsistencies (mostly whitespace usage) and the documentation (Javadoc) is poor.

2.6 Deployment View

This section summarizes the system requirements and dependencies to successfully run the Arduino IDE. Figure 7 illustrates the deployment view of the Arduino IDE.
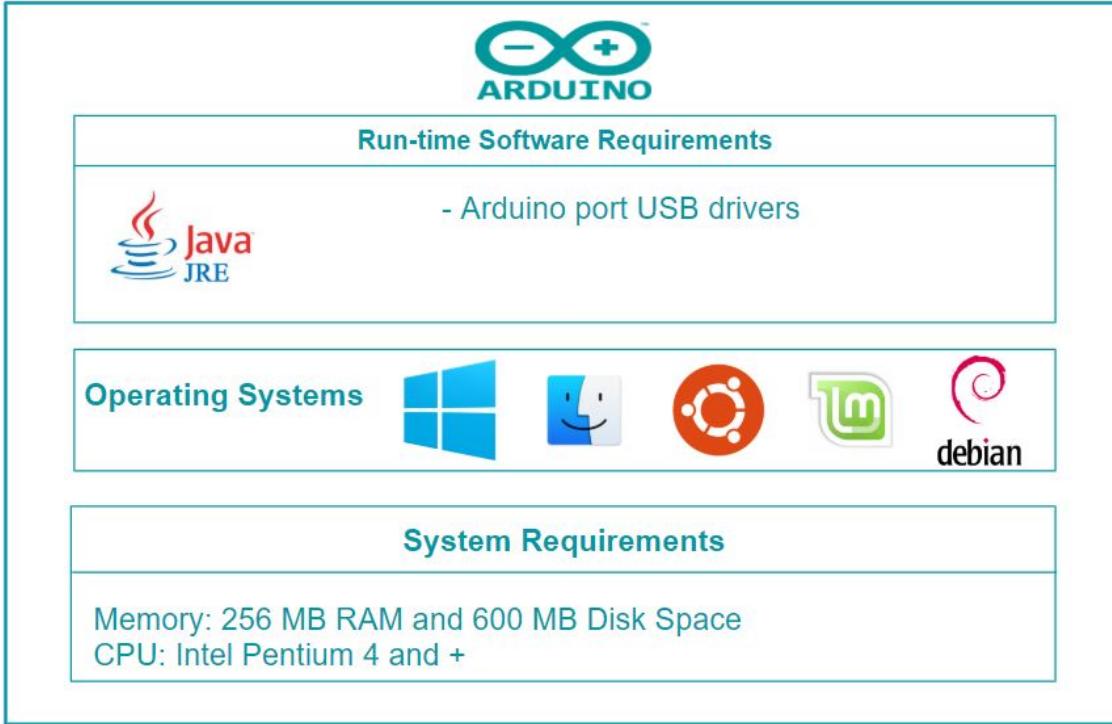
Figure 7: Deployment View of the Arduino IDE

- *Run-time Software Requirements:* Java run-time environment is an important requirement for the installation and run-time of the Arduino IDE. The Arduino IDE uses `avr-gcc` to compile the sketches written in the IDE. The Arduino IDE also requires the USB drivers to be installed to connect with the Arduino board for uploading the code from the sketch. All of these dependencies are included within the official download on the Arduino website and when building the system, which means that users don't have to install anything other than the Arduino IDE.
- *Operating System:* the Arduino IDE is built on Processing modules and Java, both of which are cross-platform. There are no official sources about the minimum version requirements for the Arduino IDE. A lower bound can be derived however by looking at the minimum system requirements for Java 8.0, the main dependency for the Arduino IDE. The following information is based on the minimum requirements for Java 8.0, which can be found on the Java website [16]. Java is supported on Windows Vista SP2 and above. For Windows 10, Java version 8u51 is required to run. For Macs, it requires OS X version 10.8.3 or above. The Arduino website states that any Linux flavor should work and that only the architecture (Intel 32 bit, Intel 64 bit, ARM) matters [17]. This is correct for recent installations of Linux, but not all older versions can run the IDE. The following list is a list of officially supported Linux distributions that are officialy supported by Java:
  - Oracle Linux 5.5+
  - Oracle Linux 6.x (32-bit), 6.x (64-bit)
  - Oracle Linux 7.x (64-bit) (8u20 and above)
  - Red Hat Enterprise Linux 5.5+, 6.x (32-bit), 6.x (64-bit)
  - Red Hat Enterprise Linux 7.x (64-bit) (8u20 and above)
  - Suse Linux Enterprise Server 10 SP2+, 11.x

- – Suse Linux Enterprise Server 12.x (64-bit) (8u31 and above)
- – Ubuntu Linux 12.04 LTS, 13.x
- – Ubuntu Linux 14.x (8u25 and above)
- – Ubuntu Linux 15.04 (8u45 and above)
- – Ubuntu Linux 15.10 (8u65 and above)
- *Hardware Requirements:* For the Arduino IDE to install and run, the system is required to have 256 MB RAM on top of the requirements for the operating system, CPU with Pentium 4 or above.
- *Network Requirements:* the Arduino IDE requires a network connection to check and download updates for both the IDE and libraries. The updates are downloaded from the Arduino website, specifically http://www.arduino.cc/latest.txt. The IDE adds parameters to the URL with specifics about the OS when getting the latest version number. The libraries specify their own URL for where to check for updates.

2.7 Technical Debt

Technical debt is a concept that represents the difference between the actual solution and the ideal solution. Choosing an easy, non-ideal solution that is often faster to implement can seem like a good solution, but it can mean that even more work has to be done in the future when working with this non-ideal solution. In computer science, this includes things such as not writing proper documentation, leaving unused or outdated files/code in the project, not creating proper abstraction layers for components that are subject to change and not writing automated tests for new code. There are several forms of technical debt, some of which are discussed below.

2.7.1 Design Debt

Design debt is created by developers making poor design choices for the system. To analyze a part of the design debt, we have used the static analysis tool PMD to identify code smells.

2.7.1.1 Code Analysis

PMD helps in identifying five types of code violations : Blocker, Critical, Urgent and Warning violations. As per PMD, there are more than 3500 violations, which made it impossible to analyse all of them. Thus, we reduced the scope to Blocker and Crititical violations, which reduced the number of violations to approximately 200.

2.7.1.1.1 Blocker and Critical Issues We have observed the below mentioned blocker violations :

- Formal Parameter Naming Convention: The parameter names provided do not follow the naming convention provided by Java Code Style guidelines.
- Avoid throwing raw exception type: Runtime exception is used rather than using Exception subclasses (e.g. ArithmeticException and ArrayTypeException).
- Variable naming convention: The variable names provided do not follow the naming convention provided by Java Code Style guidelines.

- Avoid file stream: FileInputStream has been instantiated which can lead to long garbage collection pauses.
- Constructor calls overridable method: Calling overridable methods during construction might result in invoking methods on an incompletely constructed object, which can make the debugging process difficult.

Most of the blocker issues were related to parameter and function naming conventions. In addition to the above mentioned issues, we found many critical violations like reassigning values to incoming parameters. That said, it has to be mentioned that PMD inspects and compares the code with some often used Java Code style guidelines. A tool such as PMD cannot truly determine how bad code is, it is instead used to ensure that code (and later edits to it) follows certain rules consistently, leading to a consistent code base over time. Another issue which is observed a lot in Arduino code is the lack of documentation and inconsistent code style which makes it difficult to understand all the functionalities. To summarize the above mentioned observations, it can be concluded that the code smell is quite high for the Arduino IDE project.

2.7.1.2   Issue Analysis

There are multiple issues related to the Arduino IDE. For instance, #6951 proposed that 'Ctrl+F' in mac opens the find and replace window and it takes roughly half a second for the cursur to move to the new window. This issue was raised because of a couple of commits(0d50f0b and 65103aa) related to another issue#6603. The issue mentioned was reported as a bug and it is difficult to implement the change in Java. This shows lack of end-to-end testing to find the impact of a change in other functionalities of the IDE which doesn't really reduce the count of issues as it might lead to another issue. In total, there are 110 open issues related to the Arduino IDE which still needs to be taken care of.

2.7.1.3   Historical Analysis

We have compared the code for the previous releases of the Arduino IDE(ide-1.0.x and ide-1.5.0.x) using PMD to understand the evolution of the design debt over time. It has been observed that a lot of files were removed and added during the process. As per PMD, the number of violations increased with the newer version (1.5.x). Although this comparison do not mean that there will be more problems in future assuming that both the versions didn't take code practices in account, it is important for any open source project to follow a coding style which leads to a consistent code base. A static analysis tool could have had an impact on this project as the code quality would have stayed roughly the same which is important for a project like the Arduino IDE with more than 100 contributors.

2.7.2   Documentation Debt

Documentation is of great importance in any project for both developers and users. It is very important for a project like the Arduino IDE since it allows developers to more easily find the code they want to edit and understand how the current code works. With proper documentation, one would have to read less code to understand what's happening. Proper documentation also makes it easier to compare what some method/function does with what it should do. It is equally important for users as well to understand the installation procedure and navigation of the software which can be made easy only by proper documentation. We have checked issues related to documentation raised in the past and we have observed that the type of

issues, i.e. typos and inconsistency in documentation, still remains the same. This follows from our analysis of the issues. Some examples are mentioned below.

2.7.2.1   Issue Analysis

This section looks into issues related to the documentation about Arduino IDE. There are 105 open issues related to documentation.Thus, the documentation debt seems to be really high. For instance, issue 1055 is about the improper documentation of *pins_arduino.h*, which is important to meet the needs of custom-designed boards. Another issue in Arduino documentation is the inconsistency of data. There were issues raised regarding the inconsistency in data in the website and the documentation. For example, in Issue #8086, the FPGA chip number was wrongly mentioned in the documentation.

2.7.3   Test Debt

The Arduino IDE consists of two projects: 'app' and 'arduino-core'. The 'app' project contains JUnit tests and the 'arduino-core' project contains no tests at all. However, since the 'arduino-core' project is used by the 'app' project, it is indirectly tested. In total, there are 65 tests and some of them may fail based on the Operating system(macOS,Linux or Windows) and its version. The results of running a test with instruction coverage are shown below:

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
▼ processing	6.9 %	2,758	37,095	39,853
▶ src	1.0 %	344	35,559	35,903
▶ test	61.1 %	2,414	1,536	3,950
▼ arduino-core	15.1 %	4,083	23,028	27,111
▼ src	15.1 %	4,083	23,028	27,111
▼ processing.app	5.9 %	296	4,750	5,046
▶ BaseNoGui.java	8.3 %	156	1,714	1,870
▶ Platform.java	5.4 %	38	665	703
▶ Sketch.java	0.0 %	0	628	628
▶ Serial.java	0.0 %	0	526	526
▶ SerialPortList.java	0.0 %	0	441	441
▶ PreferencesData.java	8.0 %	30	347	377
▶ SketchFile.java	0.0 %	0	316	316
▶ I18n.java	47.1 %	72	81	153
▶ SerialException.java	0.0 %	0	16	16
▶ SerialNotFoundException.java	0.0 %	0	16	16
▶ cc.arduino.contributions.packages	2.0 %	68	3,272	3,340
▶ processing.app.helpers	16.0 %	467	2,444	2,911
▶ cc.arduino	4.4 %	105	2,269	2,374
▶ cc.arduino.packages.uploaders	3.1 %	48	1,524	1,572
▶ processing.app.debug	0.0 %	0	1,221	1,221
▶ cc.arduino.contributions.libraries	32.7 %	482	993	1,475
▶ processing.app.legacy	7.4 %	76	956	1,032
▶ cc.arduino.i18n	15.5 %	156	849	1,005
▶ cc.arduino.utils	2.6 %	21	782	803
▶ cc.arduino.packages	0.0 %	0	583	583
▶ cc.arduino.packages.ssh	0.0 %	0	483	483
▶ cc.arduino.packages.discoverers	0.0 %	0	467	467
▶ cc.arduino.contributions	50.4 %	430	424	854
▶ cc.arduino.packages.discoverers.serial	0.0 %	0	372	372
▶ processing.app.windows	18.7 %	82	356	438
▶ processing.app.packages	58.1 %	481	347	828
▶ processing.app.linux	11.2 %	38	301	339
▶ cc.arduino.utils.network	62.3 %	337	204	541
▶ cc.arduino.packages.discoverers.network	0.0 %	0	85	85
▶ processing.app.macosx	78.2 %	266	74	340
▶ cc.arduino.net	85.1 %	416	73	489
▶ cc.arduino.os.windows	0.0 %	0	73	73
▶ edazdarevic.commons.net	88.6 %	218	28	246
▶ processing.app.tools	0.0 %	0	28	28

Figure 8: Test Coverage

From the above table, it can be observed that the test coverage for arduino-core is only 15.1% and for app it is only 5.9%. From the observations above, we can conclude that the testing debt for Arduino is very high.

2.7.4   Project Debt

The project debt is quite high for the Arduino IDE as there are 920 open issues and 160 pull requests still to be taken care of. Since the core developers are responsible for testing the pull requests, it becomes nearly impossible to keep track of all the pull requests on time and perform a thorough check before merging them.

2.8   Suggestions

To avoid documentation debt, we suggest the contributors to properly document their changes in relevant places. Moreover, core developers should refuse to merge the code if proper documentation is not present. In addition to the above mentioned points, there should not be any inconsistencies in different documents regarding a common topic. The backlog of issues can be a symptom of technical debt which should be considered. Similar to 'CONTRIBUTING.md' file which serves as a checklist before contributing to the repository, a file should be added to the repository regarding the code style guidelines which have to be followed for any contribution.

2.9   Conclusion

In this chapter the architecture of the Arduino IDE was analyzed using the viewpoints described by Rozanski and Woods in their book *Software Systems Architecture, Working with Stakeholders using Viewpoints and Perspectives*[5]. The *stakeholder view* identified the stakeholders involved in the project. The stakeholders were analysed using a Power vs Interest grid which led to the conclusion that stakeholders with highest interest and power include the core developers, active contributors on Github and the Arduino community. In the *context view* we identified the scope and responsibilities of the Arduino IDE and visualized it in a context diagram which shows the relationships, dependencies and interactions between the Arduino IDE and other external/internal systems, organizations, and people across these boundaries. The *context view* showed that Arduino has a well-defined scope for their system. The *development view* discussed the module organisation, common design model and codeline model in detail. The *deployment view* summarized the system requirements and dependencies to successfully run the Arduino IDE. Finally, the analysis performed in *technical debt* helped us to understand that there are lots of possible improvements to the project when it comes to code smells, documentation and test coverage.

In conclusion, Arduino is a very interesting project maintained by highly skilled engineers, but it would benefit from more active efforts to reduce the technical debt.

2.10   References

1. https://en.wikipedia.org/wiki/Arduino_IDE

2. https://github.com/arduino/Arduino/wiki/Building-Arduino

3. Comparing prototype platforms

4. Arduino Documentation

5. Software Systems Architecture, Working with Stakeholders using Viewpoints and Perspectives by Nick Rozanski and Eoin Woods

5.1. Stakeholders

6. A new era for Arduino begins today

7. Teaching, Inspiring and Empowering!

8. Arduino user groups

9. Story and History of Development of Arduino

10. Arduino authorized distributors

11. Development Policy

12. AUniter and Jenkins

13. Olander, S., & Landin, A. (2005). Evaluation of stakeholder influence in the implementation of construction projects. International journal of project management, 23(4), 321-328.

14. Arduino Policy

15. Development Policy

16. The minimum system requirements for Java 8.0

17. Guide for installing the Arduino IDE on Linux

18. https://github.com/arduino/Arduino

19. Understanding Technical Debt

20. Java Code style guidelines

2.11   Appendix

To identify the decision-making process for which PRs get merged and which PRs get accepted, we analyzed some PRs. This appendix describes the key points for that analysis. First a description of what happens for most PRs is given. After that, we will take a closer look at the kind of tradeoffs that need to be decided. We will finish with the reasons a PR can be rejected.

PRs can be divided into two categories: a *bugfix* fixes a bug, while a *feature PR* adds a new feature or changes an existing feature. Bugfix PRs are basically always considered, as a bugfix is never undesired. For the feature PRs, the core developers first consider whether they want the feature. They do this by weighing the usefulness of the feature versus how much more the feature would complicate the IDE for new users. There are few PRs that get rejected because of this tradeoff. This is presumably because contributors make an issue (or read existing issues) where they will be told that the feature is not desired before they implement the feature and make a PR for it. Nevertheless, an example of a PR that was rejected because it introduced an unwanted feature is PR #2073.

If the PR isn't rejected outright, it is iterated upon until the PR is either accepted or rejected. During that iteration, the code is reviewed and tested by the community and the core developers. Based on those reviews and tests, the following steps all happen simultaneously:

- If bugs are reported, they will often be resolved by the original author.
- Code quality issues (e.g. duplicate code) will also be reported and fixed.
- Finally, tradeoffs will be discussed with the core developers and the community.

If the PR hasn't been rejected before this point, it will be merged.

Most of these tradeoffs happen on some level for all PRs, but the choice is so clear that it doesn't really count as a tradeoff. But once every so often, there is a genuine tradeoff and the core developers need to provide guidance to the community in which way they want the project to go. The tradeoffs that we found in our analysis are:

- *Amount, severity and likelihood of being fixed of known bugs vs desirability and urgency of the PR*
  Sometimes, a known bug in a PR can't easily be fixed, or the PR has a certain urgency to it. In cases like these, the core developers must decide whether to accept these known bugs in order to merge now or to defer merging until the bug is fixed.
  In most cases, there is no urgent need for the PR to be merged, and the developers will just defer merging until all known issues are fixed, for example in PR #4519. However, sometimes it will take too long for the PR to be fixed. If the PR is desirable enough, the core developers may choose to merge the PR with the bug still unsolved. While this may seem unwise, improving software is inherently an iterative process. Waiting for each PR to be perfect would ensure that PRs never actually get merged. So, imperfect PRs get merged with the assumption that if the problems are bad enough, they will get fixed in a later PR, and if they don't get fixed then they apparently weren't that important.
  An example of a PR getting merged with existing bugs was PR #4515. It works around some suboptimal design, but it introduced a new bug on a specific board. In this case, the core developers decided that the new bug was specific enough that only a few users would be affected by it (the original author couldn't reproduce the bug), while fix in the PR affected significantly more users. For this reason, they merged the PR with the new bug still unfixed.
- *Overhead vs functionality*
  On the Arduino itself, resources are limited. This means that adding new functionality can add significant overhead. For example, PR #1803 proposed to add a functionality related to USB descriptors. But the implementation would add a constant overhead of over 800 bytes, more than 20% of the flash memory that was used without the PR. This overhead existed even when the code would not be used. Such high overhead was unacceptable to the core developers, so they required the overhead to be reduced before merging.
- *Proper code design that requires a refactor vs a workaround that works now*
  While in theory it is better to not incur design debt by allowing badly designed code, there is always a tradeoff between the cost of doing a refactor to make something work and the cost of doing it quickly now but a possibility for higher cost in the future. An example of a decision on such a tradeoff can be found in PR #2681 and PR #1803. In #2681, the core developers chose to accept technical debt and merge code that added another rule to an already complicated set of rules instead of solving it properly. In #1803, there was a simple and powerful solution that exposed internals of the IDE. While the community was in favor of it, the core developers were against it as that would mean people would start depending on those internals, which would make it almost impossible to change them. For this reason, the PR did not get merged and eventually became outdated.
- *Ease of implementation vs usability*

In general, making something easier for the user requires more effort on the part of the programmer and increases the complexity of the code. Deciding here requires a good understanding of what can be expected of the user. Because the Arduino IDE is supposed to be simple to use for inexperienced users, the core developers will almost always take the option that is simpler for the user, even if it adds complexity and development time. Examples of this can be seen in PR #1803 and PRs #3549 and #4457.

Finally, we will look at some of the reasons why a PR can get rejected.

- If the original author stops responding, the PR has a high chance of being rejected. If the code is good, someone else can copy it to their own fork and open a new PR, as was the case for PR #3549, which was continued in PR #4457.
- A PR may also be rejected if the code got merged some other way, like the previous point where someone else opened a new PR, or like PR #7029, where the code had already been merged in a rebase at some point.
- A PR is rejected when a it becomes outdated, i.e. the underlying code has been changed so much that it isn't worth the effort of making the PR up-to-date again. Alternatively, it may even not be possible in the way it was originally implemented, and implementing the feature is better done from scratch. An example of this are PR #3549 and PR #4457, which were superseded by PR #4517. The latter made a better approach available and therefor made #3549 and #4457 outdated. Another example is PR #1250, which was closed because it was outdated. @cmaglie opened a new one: PR #1726.
- PRs are rejected when a better approach becomes / is available and has been implemented. Approaches are seldomly clearly better. Instead, there will be tradeoffs: one approach is better in one area, the other approach is better in another area. Which of the two is "better" is determined by discussing the tradeoffs, often in both PRs. Then, one of the core developers decides which PR to take. An example is PR #3549 and PR #4457 mentioned above, which were superseded by PR #4517, which made a better approach available and therefor made #3549 and #4457 outdated.

Chapter 3

Cataclysm: Dark Days Ahead *(CDDA)*

Aniket Samant, Gerardo Moyers, Joan Marce Igual, Naveen Chakravarthy

3.1 Table of contents

3.2 Introduction

Cataclysm: Dark Days Ahead is a turn-based survival horror game that takes place in an post-apocalyptic world, in which there is no specific goal, and the basic aim is to survive as long as you can. The game was originally developed by Whales as open-source game under the name Cataclysm. In the days which followed, Whales brought its development to an end. The gaming community later forked the game and renamed it to Cataclysm: Dark Days Ahead (CDDA). The game is available in two graphical versions: text and graphical tile based.

It is now the property of an organization called CleverRaven. Most of the information of C:DDA can be obtained from two important sources: - The project page - The GitHub page which includes: - Issues - Project evolution during time

27

3.3   Stakeholders

By looking at the github contributions the most active stakeholders in CDDA project are *BevapDin*, *Kevingranade*, *Mugling*, *Rivet-The-Zombie*, *Coolthulhu* and *KA101*. Using the definitions of Rozanski and Woods, the *stakeholders* can be classified into:

Acquirers: *Oversee the procurement of the system or product*

The members of CleverRaven have the rights of the game. They are responsible for the budget of the project and the systems acquisitions.

Assessors: *Oversee the system's conformance to standards and legal regulations*

Creative Commons Attribution-ShareAlike is in charge of the regulations and licensing of C:DDA.

Communicators: *Explains the system to other stakeholders via its documentation and training materials*

VlasovVitaly is in charge of uploading the different languages, and most of his contributions are based on this topic.

Developers: *Construct and deploy the system from specifications (or lead the teams that do this)*

Rivet-The-Zombie and outside contributors are the project's developers.

Maintainers: *Manage the evolution of the system once it is operational*

Most of the outside contributors make small changes to the code to maintain it. *Mugling* and *BevapDin* make changes to the code to keep the game operational in the best way possible.

Suppliers: *Build and/or supply the hardware, software or infrastructure on which the system will run*

C:DDA runs with different fonts like SQUARE and GNU unifont, Catch being the suppliers.

Support staff: *Provide support to users for the product or system when it is running*

I2amroy and *Illi-kun* support the game together with outside contributors.

System administrators: *Run the system once it has been deployed*

Kevingranade is the most active system administrator. He analyzes the changes to be approved or rejected, and is considered the most important stakeholder of the project.

Testers: *Test the system to ensure that it is suitable for use*

These are the gamers and CleverRaven members. They detect bugs in the game.

Users: *Define the system's functionality and ultimately make use of it*

System administrators are the users since they run the game.

The definitions given by Rozanski are really accurate, and there are also other types of stakeholders such as **customers** who treat the software as a product that passes through their systems. In this case they would be the gamers that identify a bug in the game and inform about it in GitHub. Other possible stakeholders are the **competitors** - C:DDA is a game where new worlds can be created like in Minecraft; however there is a huge quality gap between them, probably due to the latter being much more popular. Nevertheless, there are more type of stakeholders which are not contained in this project. For instance, C:DDA doesn't have any **sponsors**.

Stakeholders are an important part of any architecture - when a change needs to be made, a stakeholder analysis helps address that change to the people that can actually help. This analysis saves a lot of time in the future when an update is needed.

3.3.1 Integrators

Three stakeholders manage the changes made in the project. These are *BevapDin*, *Kevingranade* and *Mugling*. They are supposed to maintain the coding style. Also, making really small changes is not accepted but fixing spelling mistakes is. Bug fixes are extensively verified before they are accepted.

Solving an issue is not a trivial task. Even thought the issue is solved in terms of programming, there are some factors that have to be taken into account like the writing style and not adding more bugs instead.

3.3.2 Relevant people to contact

After analyzing the project's stakeholders, four contributors have been found to have the biggest understanding of the project code and its goal. These are:

1. Kevingranade
2. Rivet-The-Zombie
3. BevapDin
4. Mugling

3.4 Context Viewpoint

Apart from the stakeholders described in the previous section, the project relies heavily on some external entities as well for successful development and release, and those are dealt with in the subsections that follow.

3.4.1 System Scope and Responsibilities

C:DDA is a video game designed to run on a laptop or a desktop computer. C:DDA gives the users an uncompromising survival scenario, and it also helps gamers in developing problem-solving abilities. This game only has the responsibility of entertainment for users, with the highest quality achievable.

3.4.2 Distributions

The game is available in the form of: - *Stable Releases* (Windows and Linux platforms) - *Experimental Builds* (Windows, Linux, and OSX platforms)

Moreover, the Experimental has two different types: - *Console* (referred as Terminal): the game is run like a console application, with graphics being represented using ASCII characters, and - *Graphical* (referred as Tiles): the game is run with additional graphical elements.
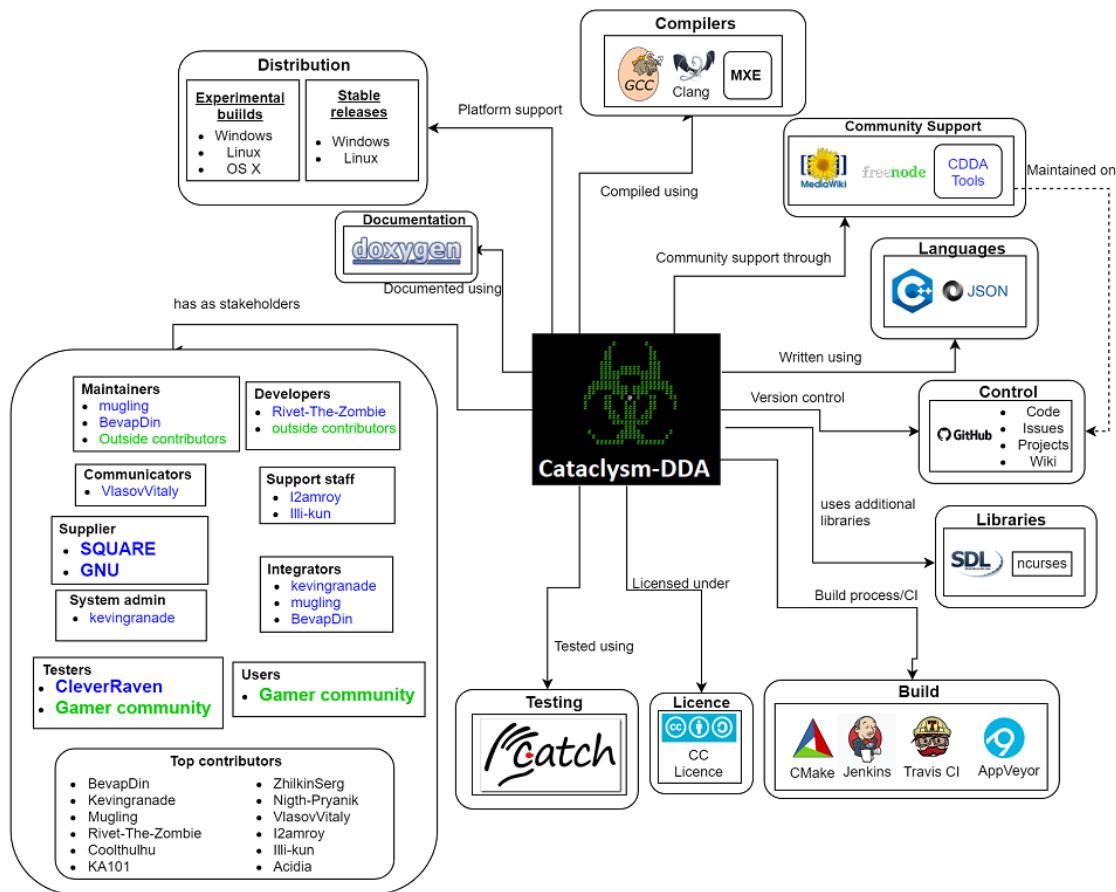
Figure 3.1: Context schema

Android builds are also available, and are available on the game's Releases page.

3.4.3 Development and Contributions

3.4.3.1 Development

The game is mainly written in C++, and most of the data (mods, items, NPCs, etc.) is present in the form of JSON files. Additional scripts (lua, Python) are also used in the build process. i18n support is also added to support multiple languages. SDL and ncurses are also used.

3.4.3.2 Compilers

The game can be compiled using the following: - GCC - Clang - MXE

3.4.3.3 Version Control

The project is hosted on GitHub and the *Code, Issues, Wiki, and Projects* pages are used for their respective purposes. Any new contribution has to raise an issue or post a pull request.

3.4.3.4 Build Processes

CMake is used for development purposes although it can build the project unofficially. For official builds, the Makefile in the project's root directory is used.

The project uses Jenkins CI for its daily builds, and also uses AppVeyor and Travis CI for continuous integration of pushed code.

3.4.4 Testing

The Catch unit test framework, based on Boost libraries, is used for performing unit tests on the code.

3.4.5 Community Support

The project has a dedicated website, a dedicated discussion forum, a series of web tools hosted through another GitHub project, and is also supported through an IRC channel. There is also a wiki page for all the game related information.

3.4.6 Documentation

Doxygen is used to document the entire codebase, in addition to having documentation files in the GitHub repository for contribution guidelines, compiling, etc.

3.4.7   Licensing

The project is licensed under the Creative Commons Attribution-ShareAlike 3.0 Unported License.

3.5   Development Viewpoint

3.5.1   Introduction

CDDA is a vast project with approximately 251k lines of code (as reported by SonarQube). The structure of the files is not considered to be the best given the way everything is structured (more depth analysis done in the later sections). All the source files are contained at a single level making the code management very difficult. To get a clear understanding for the development view, efforts were put to segregate the monolithic module into components to enable clear visualization of the underlying functionality.

3.5.2   Module Organization

Most of the information is taken from the documentation [1]. Each component defined in the following diagram is presented in the code in the form of classes. These components are categorized into layers with respect to their functionality and level of operation.

core: This layer is mainly responsible for setting up the environment of the game, initializing all the components that are required, and orchestrating the complete game. Every single layer given in this diagram either directly or indirectly communicates with the core layer. The core layer also in turn communicates with another layer called common layer which is mainly responsible for other supplementary functionalities.

common: Contains all the modules that are going to be processed by all other modules present in other different layers. All components in this layer are modified as part of their feedback to an action that took place as part of interaction in the game.

Vehicle Specifics: This layer contains the components that support vehicle functionalities and interface to the modules present in the core level.

vehicles: This layer contains the components whose instance are created from JSON files to make the maintenance easy. Each of these components is mainly responsible for ease of mobility of the player.

creatures: Each of the components under this layer is the living being that is either harmful or harmless. NPCs (Non-Playable Characters) could include doctor, pets, etc., while monsters include zombies, vampires, etc.

possession: Contains all the components that are modeled as behavioral attributes for the player and the NPCs. This layer contains all the components that consist of various kinds of attributes the player possesses which are represented in the form of classes to allow flexibility in interactions.

item: This layer contains all the components that are instantiated dynamically during gameplay. Each of the components defined in this layer is supplied in the form of JSON files to enable maintainability.

---

[1]Module documentation, http://dev.narc.ro/cataclysm/doxygen/pages.html
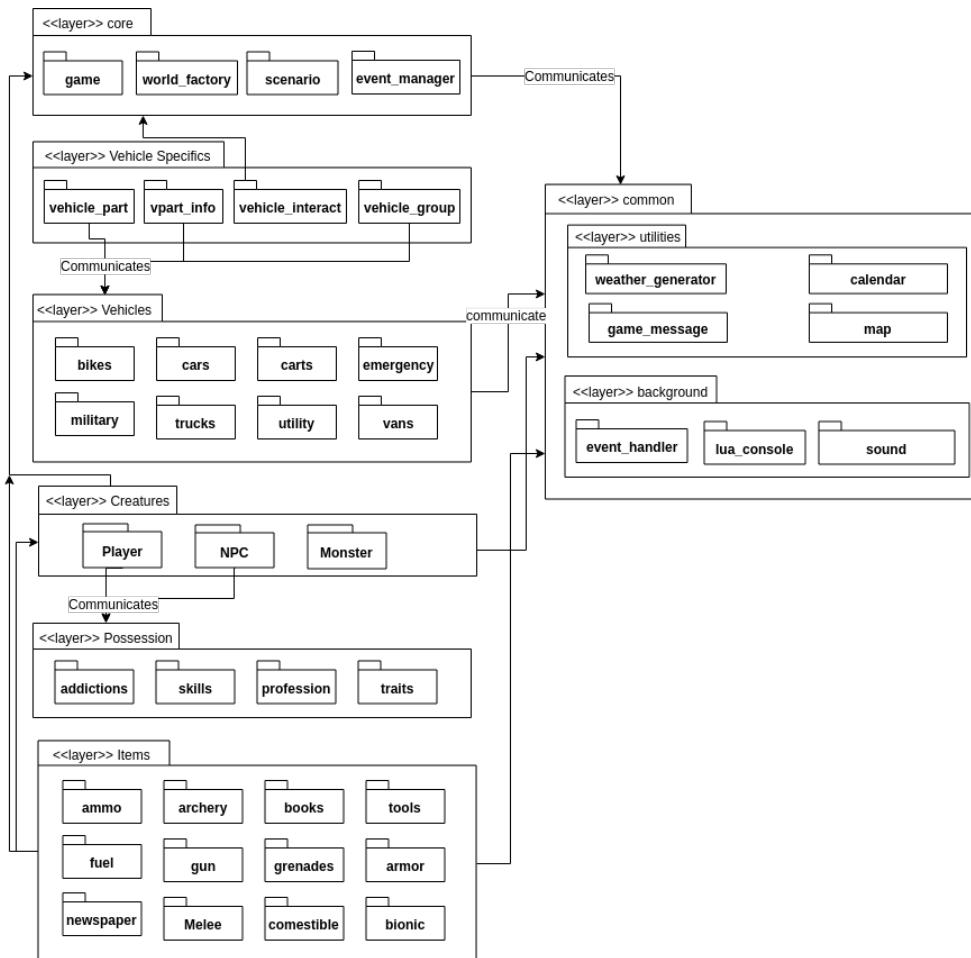
Figure 3.2: Module Structure

3.5.3   Common design models

As a way of reusing code, Cataclysm-DDA uses a library in order to render the game's graphics, and the library is chosen at compilation time: - **ncurses** if the game has to be rendered in a terminal - **SDL** if the game is directly rendered in its own window with sprites

3.5.3.1   Standardized design

Cataclysm-DDA accepts all types of contributors. However, they have to respect some rules when contributing in order to be accepted.

There is a `CONTRIBUTING.md` file [2] that states the rules that have to be followed. When contributing there is a commit template that is specified in the `.gitmessage` file [3] and that all the developers must follow. Moreover, when creating a pull request the first line of the description needs to be a summary line with a specific format that will be later used to create the release changelog.

Also, to ensure that all the code has the same style there are some styling rules for C++ defined in the `CODE_STYLE.md` document [4].

In the `JSON_INFO.md` document [5] the contents for all the JSON files are described and the syntax that should be used in the files, there's a total of 96 JSON files specified. The files are classified into general files describing some game messages and basic items, item files adding objects to the game, requirement files that set the required items for crafting some objects and vehicle files that describe different types of vehicles and how they are used.

3.5.4   Codeline Organization

In this section it is explained how the code is managed, tested and built.

The source code is structured in a "flat" form, in that there are no subdirectories within the source directory, and it is difficult to understand purely by looking at the code structure (file names) what the various responsibilities of the files are (for instance, `fungal_effects.cpp` or `melee.cpp` provide no idea what the meaning could be).

3.5.4.1   Testing

The testing system is mostly automated. Even thought testing is an important part of software Cataclysm only covers almost one third of the code with tests.

3.5.4.1.1   Testing framework   The project uses Catch as the testing framework. All the tests are located in the `tests/` directory and every test file tests a different file from `src/`. Not all the code is covered by the tests since the total coverage is around 29%, this measurement is done through Coveralls.

---

[2]Cataclysm-DDA rules for contributors, https://github.com/CleverRaven/Cataclysm-DDA/blob/master/.github/CONTRIBUTING.md

[3]Cataclysm-DDA git commit message template, https://github.com/CleverRaven/Cataclysm-DDA/blob/master/.gitmessage

[4]Cataclysm-DDA code style rules, https://github.com/CleverRaven/Cataclysm-DDA/blob/master/doc/CODE_STYLE.md

[5]Cataclysm-DDA JSON file contents, https://github.com/CleverRaven/Cataclysm-DDA/blob/master/doc/JSON_INFO.md

3.5.4.1.2 Continuous integration Continuos integration is a development technique used when many developers work together changing parts of the code. It is usually used every time that there's a pull request or a release in order to check the new changes and look for errors as soon as possible. C:DDA uses Travis CI to check Linux and MacOS builds and AppVeyor to check Windows builds.

Through the CI, they make sure that the code compiles properly and then they execute the tests to check that the new changes do not break any existing code. There's also an additional test that builds the code and deploys a new build if it is merged, it also checks for a properly formed pull request message. They use Jenkins in this last check.
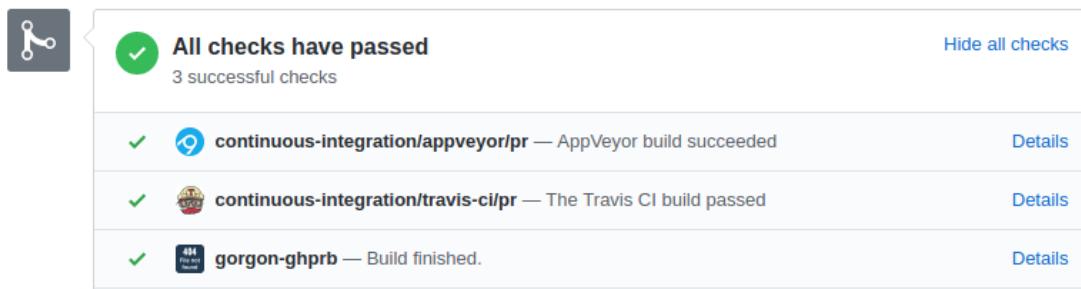
Figure 3.3: Continuous integration

3.5.4.2 Building

Cataclysm has major versions (`0.C` and `0.D`) but the release daily builds. This means that every day when a pull request is accepted this is uploaded into the releases page.

The building process is done through GNU make. The makefile defined has different options to create a *release* or a *debug* build that can be defined when calling make. The makefile also supports different platforms so from one platform it can be build for other ones. In general from Linux it can be compiled to Windows, Mac OS or Android as a cross-compilation option.

3.5.4.3 Release

The releases of the game are automatically created and uploaded into the release website [6] every time that a pull request is merged by using a Jenkins script.

3.5.4.4 Instrumentation

Another way that the developers use to know how the game is behaving is through instrumentation. This is done through debug builds that print messages to the console that later can be used to understand how the code was behaving when an error ocurred. This is the default build method of Cataclysm-DDA and to disable it the flag `RELEASE=1` has to be added when calling `make`.

---

[6]Cataclysm-DDA Linux Tiles releases, http://dev.narc.ro/cataclysm/jenkins-latest/Linux_x64/Tiles/
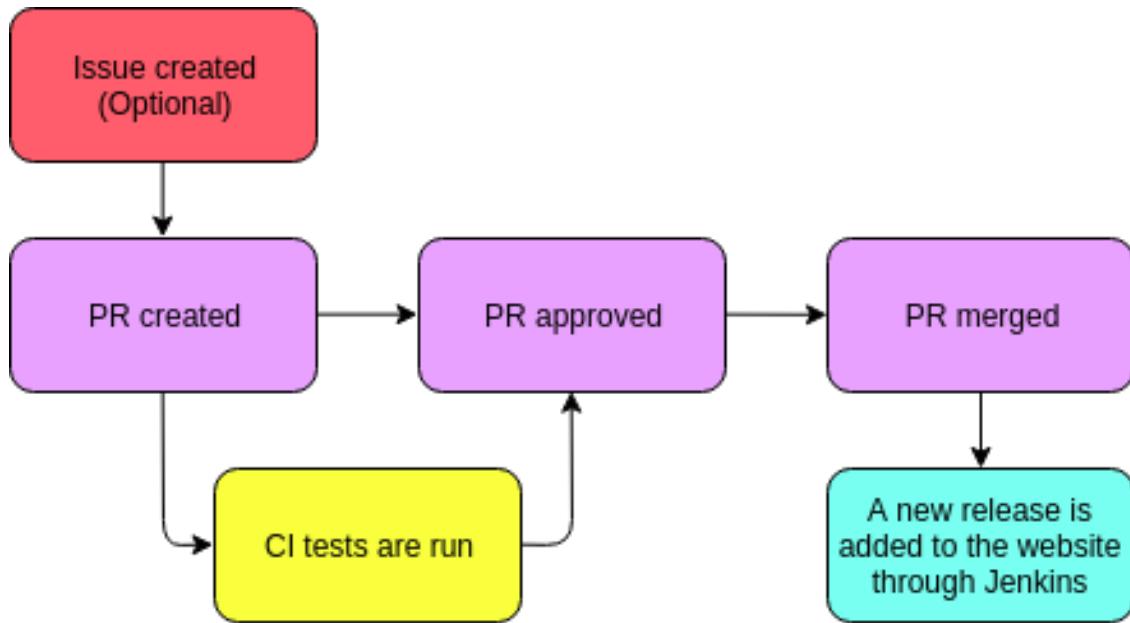
Figure 3.4: Releases

3.6   Functional Viewpoint

As part of a new perspective, we chose to add this viewpoint as it is more relevant in terms of the game structure. The game is turn-based in which the player makes a move (turn), the NPCs subsequently make their own moves, and then the control is returned to the player ad infinitum. This sequence goes on till the user decides to quit the game (or is killed).

Based on this idea, the game is run in the form of a single `while` loop which calls several functions related to each character's turn and their interactions related to other components. From a high-level perspective, the functioning of the game can be represented graphically as seen below

3.6.1   Modules

Since the game is supported on multiple platforms (Android, Windows, OSX, and Linux) with multiple builds (Tiles and Curses), some of the platform-specific functional elements are compiled accordingly (using `#ifdef`). Moreover, there are mods available which are loaded at run-time depending on installations.

3.6.2   Game Elements and Interactions

This section is based on manual code inspection to understand what the various elements do and how components interact.
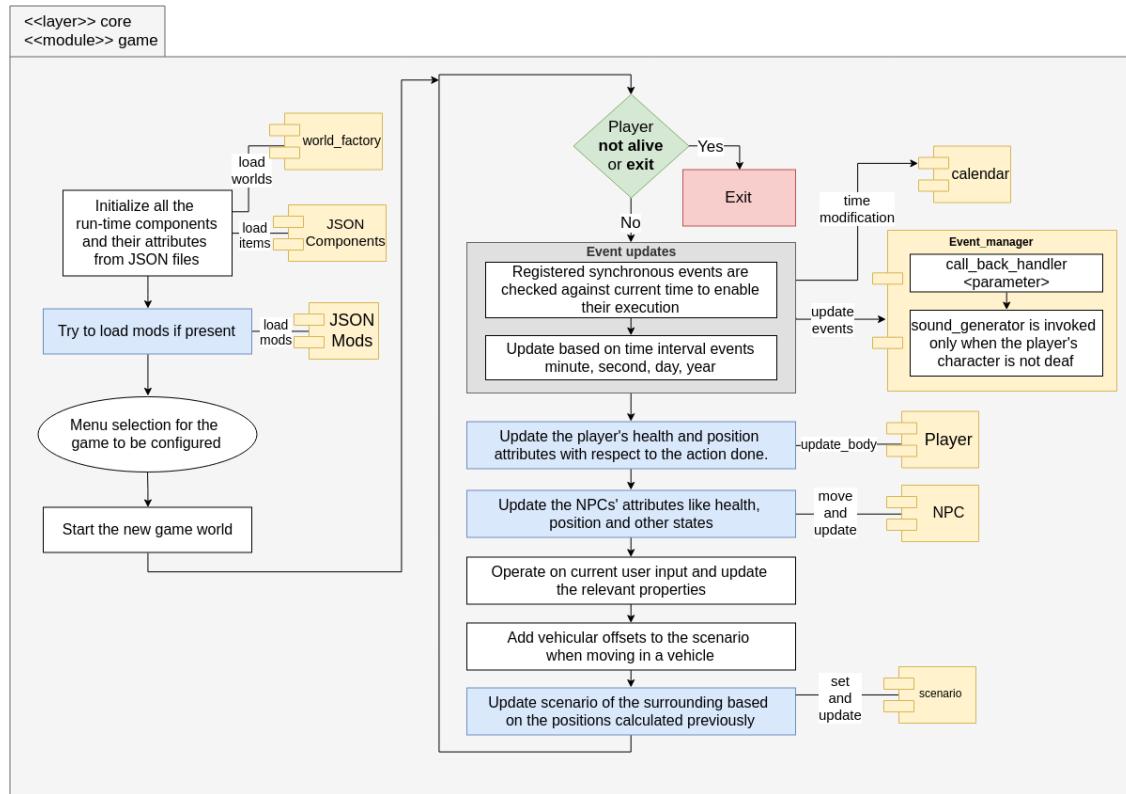
Figure 3.5: Context schema

3.6.2.1   Main flow

- A `game` instance is instantiated during the start of the game with a *unique_pointer*, to make sure there can only be one instance of the game running.
- As stated previously, the game runs in the form of a `while` loop whose exit is handled if the player quits the game.
- The class `game` consists of all functions required to run the game and display the UI.

Below are some important modules that are crucial during the run of the game.

3.6.2.2   game

The `game` module's `do_turn()` function is called in every `while` loop iteration in the main program. This function internally performs a lot of actions in the form of calling functions defined in the game class itself. A high-level summary of what goes on in the loop is as follows (and also the associated interactions, if any): - Checking if the game is over - Checking if the NPCs have changed their states - Event-based updates (like weather changes, healing process, etc.) - Auto-saving the game - Adding new NPCs randomly - Interaction with the `user` object to process user's input actions and other miscellaneous activities (dealing with sleep, healing injuries, etc.) - Handle `vehicle` actions if applicable - (Re-) drawing the visible map - Processing sounds

3.6.2.3   player and NPC

The player of the game is modelled as a `user` object with which interactions occur. Player actions are accumulated in a data structure and then the `game` objects processes the actions one at a time. The user can quit the game at any point, or can be killed - each of which causes the game loop to be broken and the game to end. The game polls for player's input in the `while` loop under the function `do_turn()`.

The NPCs are modelled similar to the user object and their interactions are with the `game` object. Sounds and actions are processed in every `do_turn()` iteration.

3.6.2.4   Event_manager

Each components during the their initialization, register their events that should occur periodically. Few such events could be changing the daylight in the game every thirty minutes, healing wounds of the character (player or NPC) by a certain amount every one hour, changing the season in the game every three months. During every iteration of `do_turn()`, the registered events are checks for triggering them individually to change their state and later update their effect on other components.

3.6.3   Summary of functional viewpoint

Since this is a single-threaded game in which everything (game actions, sounds, graphics) gets processed in one loop, the interactions between the various elements are simple and can be understood by going through the code step by step (there's no scope for race conditions). The inter-element interactions are mainly the

ones between the character objects and the main game object instance, and the rest of the interactions are miscellaneous functions related to game startup, load/save, etc.

3.7 Technical Debt

3.7.1 Introduction

This section deals with the technical debt seen in the project. Generally speaking from a high-level perspective, since this is an open-source *game* project with no major consequences of bad code quality on the end users (the gaming community), it is clearly seen that the code is not maintained up to a good standard. It could well serve as an example of how a software project could end up in a mess if not maintained from the very beginning, as will be detailed in the forthcoming subsections.

3.7.2 Metrics

The game is written mostly using C++ 11 with JSON files for data fetching, and hence a starting point for analysis is to calculate code metrics particular to C++. Two main approaches were followed to evaluate the technical debt of the project: - Static code analysis tools (Sonar-cxx plugin for SonarQube, CppDepend, and CppCheck) for code metrics - Manual inspection of the code to get an understanding of the development style and patterns

Though the project's repository provides a direct Visual Studio solution consisting of all the source code, its analysis using VS proves to be cumbersome. The Visual Studio 2019 blog provides an insight into what can be expected from VS 2019, and more specifically a good C++ code metrics analysis experience. VS 2017 does not provide intuitive tools for it, and hence wasn't be used for analyzing the source code.

SonarQube was attempted to be used to analyze the project for metrics, but using the standard SonarScanner installation (through SonarCloud) for C++ failed to work, and hence an open source SonarQube plugin, Sonar-cxx was used instead. The following analysis is provided:

Figure 3.6: SonarQube analysis

CppCheck provides more insights about the code, as seen below.

We see that there are very few *errors* in the code, and most of the reported problems pertain to styling issues and warnings (which must not be overlooked, ideally).
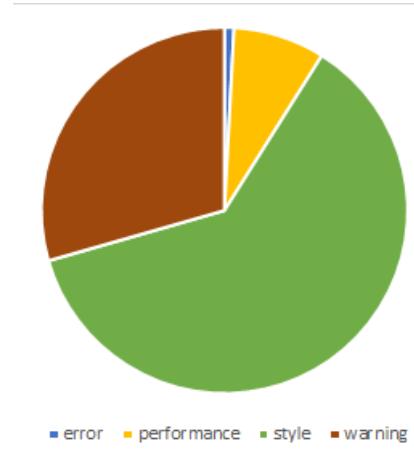
Figure 3.7: Types of problems
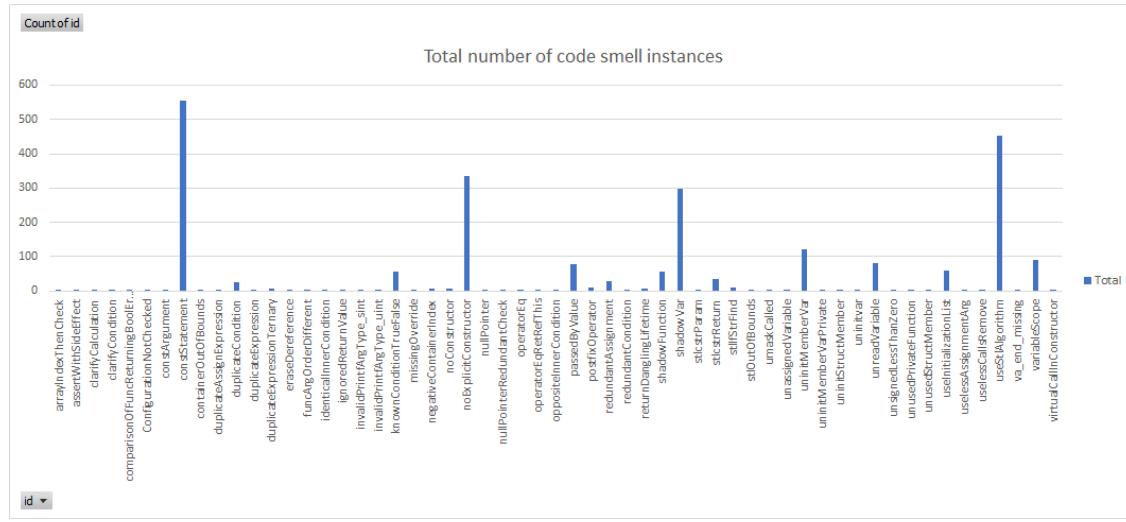
Figure 3.8: Code smells

In order to understand the issues related to code smells, performing an in-depth analysis yields the following data:

A pattern is observed in the 251k lines of code calculated using the tools - some code smells stand out more than others, and the following set of code smells shows up prominently: - A lot of lines of code start with a string constant - error prone in terms of assigning string values while writing code - Not using the `explicit` keyword in constructors with a single parameter - Shadowing of outer variables by inner variables in many cases - Lack of use of the C++ STL which would otherwise be more efficient than raw loops - There is a duplication of nearly 1.5 % in the source code - it translates to a lot of duplicated lines in absolute numbers and could be error-prone

3.7.3 Code debt

As previously stated, the source code is structured in a "flat" form, in that there are no subdirectories within the source directory, and it is difficult to understand purely by looking at the code structure what the various responsibilities of the files are. If the code had been structured in a logical directory format, it would have been much easier to understand and maintain. Moreover, the cyclomatic complexity of several functions and methods is very high given the number of `switch case` and `else if` blocks present. These are very clear signs to show that the code is very difficult to maintain.

The bus factor of this project is very high (most of the code is gated by Kevin Granade) and hence there is no guarantee of it being stable in the long run, unless the core development team is distributed across more developers. This is the most worrisome factor for the project, and definitely needs to be handled to allow the project to continue being developed from a long-term perspective.

In terms of *SOLID* violations, the following can be said: - **SRP** is the most repeatedly violated principle in this project and it can be spotted easily. There exist several classes that have multiple responsibilities, and they could easily have been split into smaller ones. - The rest of the principles can be commented on from a high level (for instance, the LSP is most likely not violated since the class structure is flat, similar to the directory structure; the ISP is expected to be violated given the number of `virtual` keywords present in classes; the DIP is probably not applicable given there aren't interdependent *modules* loaded by the code) but their violations don't stand out in comparison to the SRP's.

3.7.4 Testing debt

Any commit made to the source directory is through Travis CI and AppVeyor, and hence unit tests are performed on the changed code before being allowed to be merged into the master branch. However, the test coverage for the project is quite poor, at 29%, and hence there isn't a good guarantee that the tests written would prevent issues from arising during runtime, even if the code compiles successfully and passes all ASAT tests.

The testing debt can be reduced by writing new unit tests and ensuring that any new commit that is made has associated unit tests with it. However, it puts the burden of ensuring test cases to be written on the contributing developer and does not seem to be good practice. An overhaul of tests is much needed for the project. Moreover, the presence of "mega-classes" (for instance, the file *game.cpp*) in the source code makes writing good tests virtually impossible.

3.7.5   Documentation debt

Doxygen is used to document the source code so as to help new developers come up to speed with regards to understanding the responsibilities of the various classes and methods. However, on going through the codebase, it can be seen that there is quite a lot of code that is not covered in the documentation, and moreover there is a lot of inconsistency in terms of style and volume of documenting (some code snippets are under-documented and some have unnecessary documentation). Though it is a good practice to follow a standard style of documentation and the developers of C:DDA have even provided guidelines for it it is clearly seen that good documentation practices aren't enforced in the project. In fact, they also recommend that new developers may require to document the internals of implementations since "many classes in Cataclysm are intertwined." This shows that they are aware of the technical debt present in the project already.

Such documentation debt is only expected to grow over time unless code reviewers enforce good documentation practices by rejecting pull requests for poorly documented code.

3.7.6   Defect debt

There exist lots of code snippets wherein comments pertaining to "TODO:"s and "hacks" are present, so we can clearly see that defect debts are quite common in this project. Implementing the corresponding code in the form of marking issues and through new contributions would reduce the defect debt greatly.

3.7.7   Evolution of technical debt

On going through the commit history of the source code, it can be seen that right from the very beginning the files have been organized in a flat structure, and hence the technical debt in this regard has simply grown over time - it seems unlikely that the structure would change, and a good understanding of the codebase would only be with the original developers.

Defect debt has also risen over the years, as can be seen below. These numbers are from commits made at some points in the years mentioned.

Given that the game runs on ideas for new features like weapons, food, abilities, etc. (for instance, see PR #29366), defect debt is expected to increase with the increasing popularity of the game.

Based on the current trend of not enforcing the writing of covering code tests, testing debt is likely to grow in size unless testing rules are enforced for all future commits.

Documentation follows a similar trend as testing, as quite a few discussions take place in the form of comments between contributors on GitHub, but a follow-up of documentation in the code not enforced.

3.7.8   Managing technical debt

Though the project suffers from a lot of issues related to technical debt, it can certainly be managed. The following points can be taken into consideration: - *Adding more developers to the core team*: As of now, only a handful of core developers know the game's core functionality properly. The core team will *have* to be expanded to keep the project active in the long run - *Adding missing documentation*: The current code
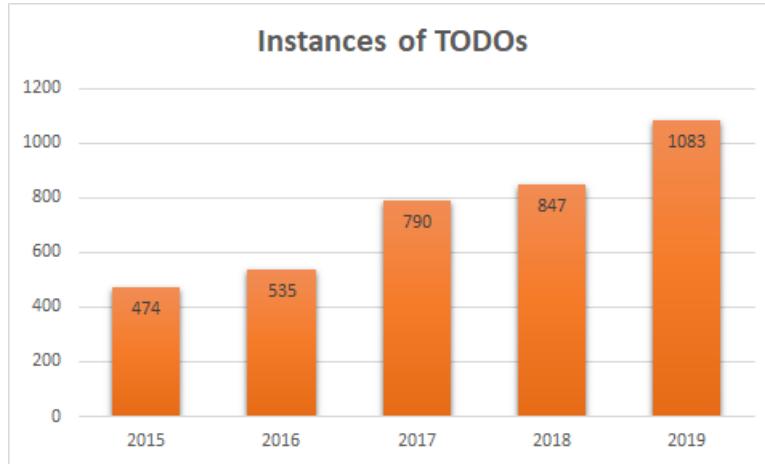
Figure 3.9: Evolution of defect

is not documented adequately. Enforcing good documentation practices should be a must for all future submissions and the reviewers need to be strict about it. The developer team is aware of this issue, as is seen in the form of a dedicated GitHub project. - *Refactoring major parts of the code*: Ambitious as it may sound, most of the code can be refactored by forking into a dedicated project and having a set of developers working to split the existing mega-classes into smaller chunks, and creating a logical directory structure (the existing files can be moved to subdirectories like *game*, *item*, *player*, *NPCs*, *map*, etc.)

A project like Cataclysm:DDA runs on ideas provided by developers and non-developers alike for new features (weapons, food, etc.), and it is only expected that a good volume of defect debt will persist. Those features may require a lot of rework to be backward compatible, and adding them with the "TODO"s would probably be the fastest way to roll them out. This is a structural issue that cannot really have a solution in a project of such a dynamic nature, and it can only be managed, not eliminated.

3.8 Conclusion

This chapter was a result of analysis of the project Cataclysm:DDA. The game could be appreciated for its complexiety, which is obvious based on the fact that it doesn't use any external game engines. The game is fully written in C++11. Unlike other open source projects where sometimes there are big companies supporting them, as this game is mainly for entertainment, the stakeholders are mostly players that play the game and also take multiple roles at developing it.

To understand the vast architecture, UML diagrams of the whole class structure and their interactions were analyzed. One major downside of the project was that it has no proper documentation of the architecture and functionality. There were a lot of efforts put to manually analyze the code through instrumentation and debug builds. On the technology front, the project has been using tools (like CI and testing) to make the development process smoother and maintainability easier.

Analyzing the project's PRs showed the denial of new features by the maintainer. However, any kind of anomoly in the functionality is taken care by a few lead contributors. This, being a modular game, opens up

space for other gamers to add MODS they wish, leaving the core of the game untouched.

With roughly 200 components in a flat structure in the game, it is onerous for one to understand the architecture and fix issues. To exacerbate the code organization, a few of the classes are humongous in length. In short, the project violates many SOLID design principles.

Upon analyzing, we suggest that it is high time that the architecture be split into modules before the maintainability goes off hand.

3.9 Annex A - PR Analysis

3.9.1 Merged pull requests

3.9.1.1 1. Vehicle propulsion overhaul - #19275 - (November 2016)

This pull request changed the propulsion component and added engine gears. Previously the users were creating too powerful vehicles that went at non-realistic speeds. Some contributors did not like it since some countries usually drive automatic cars and that would confuse players. Finally, after some bug fixing, the pull request was merged. However, the changes were reverted by the owner because it broke other components.

3.9.1.2 2. Better handling of item templates - #19583 - (December 2016)

Related to bug #19566 introduced in PR #19058 and was created one month after introducing the bug. This PR fixed a pointer issue introduced in the previous one and also improved performance. Most of the discussion went around whether to use or not the `assert` macro in release builds. Finally it was decided not to use the macro and the PR was merged.

3.9.1.3 3. [READY] Light attenuation based system for FoV and dynamic lighting - #12290 - (May 2015)

Addressed many issues related in how the light and the vision worked in the game which were part of the legacy code. The discussion first was about what should the player see during the night and then it continued with bug fixing and finally it was merged.

3.9.1.4 4. More Survival Tools Series 6 - #12405 - (May 2015)

Added more objects to the game and balanced some already existing ones in order to make the game more realistic. The discussion was focused in small bug fixes, changes that may nerf too much the game balance and how some developers play it differently.

3.9.1.5   5. Rebalance aiming and skill training - #18020 - (September 2016)

Balanced how the aiming and shooting skill training worked. It was mainly focused in changing the amount of dispersion that a bullet could have. Some developers found the new shooting range to be too short. After agreeing on a new range and some bug fixing it was merged.

3.9.1.6   6. [DONE] [CR] Field dressing corpses aka Butchery overhaul - #24480 - (August 2018)

Changed the mechanics for butchering a prey, it was related to the discussed issue #24145 that asked for development in food preserving methods. Some developers did not like the new idea but others argued that it made the game more realistic. After reviewing the code it was merged.

3.9.1.7   7. Skill training overhaul - #11695 - (March 2015)

It redesigned how the skill leveling worked from linear to exponential and removed a level cap. Everyone seem to agree with it but some developers wanted a level cap back as this topic had been previously discussed. After adding the level cap again and some bug fixing it was accepted.

3.9.1.8   8. [CR] CMake: update rules; add Linux install targets; add support for dev-builds - #11970 - (May 2015)

Updated the cmake build rules which had not been updated because cmake was not the official build system. The PR author did not have a windows machine so most of the discussion was with other contributors that could test the configuration on Windows. After making sure that it worked in every platform it was merged.

3.9.1.9   9. Zombie burner - #23729 - (June 2018)

Introduced a new zombie creature with a flame thrower. Some developers liked the idea while others thought that fire was to difficult to handle for the player and thus this creature was overpowered. After removing some of the zombie's abilities the PR was merged.

3.9.1.10   10. Use %zu for printf-style formatting of size_t values. - #19339 - (November 2016)

Updated a `printf` flag to the C++11 style. Some bugs were fixed and it was merged. However, they later found out that some compilers were not properly supporting the new flag and the changes were reverted while they were updating the compilers. Finally it was re-introduced again in #19488.

3.9.2   Rejected Pull Requests

3.9.2.1   1. Sidebar info3 - #27809 - (January 2019)

Suggested reordering and alignment of the sidebar element which included many changes to the C++ source code that dealt with the sidebar. Contributors raised concerns that aesthetics should be given least preference.

While the author argued the changes might help beginners, contributors pointed out that it could be more obvious to anyone as time progressed. This PR was superseded by another PR created around the same time leading to closure.

3.9.2.2   2. Sidebar to panels - #27439 - (January 2019)

Proposed a UI overhaul, in which the sidebar would be fragmented and involved modifying many JSON files and other source files concerned with the sidebar. Contributors raised their concern of having different text field widths for various languages. Others pointed to the unused space in the sidebar region with this PR. This PR was closed as a result of dissatisfaction among the community and the owner.

3.9.2.3   3. Reduced HP regen to 70% current value - #25628 - (September 2018)

Dealt with the regeneration of health calling the current too high. Proposal was to be reduce it to 70%. Issue related. Involved changes to the JSON files that handled the regeneration rate. The author wanted injuries to be more punishing than simply being healed quickly. The author asked for more suggestions and corrections to his comments. The author failed to give a convincing justification and resulted in closure of the PR.

3.9.2.4   4. Foundation for implementing medieval mod shields - #14711 - (January 2016)

Proposed to use off-the-hand shield which consisted of various modification to both source and JSON files. The discussion dealt with core C++ code with which the author was not confident enough. After investigating, it was realized there needs a lot of work to be done. Hence, the author split the PR into multiple PRs and closed the existing PR unmerged to favor more focused development on others.

3.9.2.5   5. Enable frostbite, remove cold damage - #18401 - (September 2016)

Proposed ways to bring frostbite into the game and changes included only JSON files relevant to armour component. Contributors brought some challenging use-cases to the author and suggested making the frostbite a gradient level. By the end of the discussion, contributors suggested that doing this would re-enable the dead-code. Hence, to merge, the dead-code was needed to be analyzed. To which the author felt he was not competent and led the author to close the PR himself.

3.9.2.6   6. Mining Mod - #18459 - (September 2016)

Author proposed to make the occurrences of metal ores more frequent and contained changes to a few JSON files. A contributor gave suggestions and offered some help in rewriting parts implementation he suggested. The owner suggested a backwards sketching and due to the lack of this, the PR got closed. This was addressed two years later by another contributor.

3.9.2.7   7. Improbable weapons mod - #18332 - (September 2016)

Proposed to eliminate some improbable weapons and had changes to only JSON files related to weapons modules. Contributors pointed that the fun in the game was usage of improbable weapons and bringing them down would only bring dissatisfaction among the players. For the amount of changes needed, after realizing the efforts were not worth and might break something else more functional, leading the author himself to close.

3.9.2.8   8. Reload using magazines - #14949 - (January 2016)

Focused on bringing more tools to the rifle magazines. This PR seemed to be very important at the very first sight and also more obvious and contained significant changes across JSON files and source files that dealt with ammo. Owner suggested that this be added as a mod first. While initially merged there were a few functional flaws with the display of ammo and lead to reversion.

3.9.2.9   9. Use SDL_gpu library for faster SDL rendering - #19412 - (November 2016)

Was meant for fixing an issue where the game was slow. It proposed on using GPU. and involved changes to the build system and source files relevant to the graphics. The author pointed a caveat to his implementation. In the midst of the discussion, the owner commented that it could be a deal-breaker. The owner affirmed that unless this change got tested on a software-rendered build, this PR would be left unmerged.

3.9.2.10   10. Increases blocking ability of smaller shields - #21503 - (July 2017)

Wanted to increase the blocking capability of smaller shields and included changes to JSON files related to the shield component. The author found the idea of increasing blocking ability for smaller shields not right in its essence. As the owner was ignoring the changes the author himself closed the PR.

3.10   References

Chapter 4

Eclipse Che

Gerben Oolbekkink (gjwoolbekkink), Ana Šemrov (asemrov), Yorick de Vries (yorickdevries), Rukai Yin (ryin)

4.1 Introduction

Eclipse Che is an open-source workspace server and an integrated cloud-based browser IDE, with the aim to offer a flexible, extensible, distributable and zero-install software development environment.

Figure 4.1: Eclipse Che logo

To put the project in a historical context, the eXo Platform first started with the idea of a cloud-based IDE in 2009 [1]. After three years of development and a substantial increase in popularity of the software, an investment was obtained to build a highly customizable commercial cloud development product and Codenvy was founded as a standalone business [2].

In 2014 Codenvy donated the intellectual rights to the Che kernel to *Eclipse Foundation* [3] and Eclipse Che (named after Cherkasy, Ukraine, where most of the development takes place) was announced as one of the top-level projects of *Eclipse Cloud Development* [4]. Codenvy then became a strategic member of the Eclipse Foundation, taking a board seat[5]. Nowadays Codenvy still bases its flagship product on Eclipse Che and was acquired by Red Hat in 2017[6], making Red Hat the biggest contributor to the project.

The Eclipse Che team believes in the future of cloud development. "Cloud development has the potential to turn organizations into lean, continuous operations and enable teams to release more frequently and ship faster", said Tyler Jewell, the Che project leader and former CEO of Codenvy[7]. The project thus aims to provide a cloud development solution with all the technologies, platforms, and protocols necessary in order to ease the path to global adoption of cloud development.

Developing in the cloud relies on development environments that can be built and set up easily, with one-click, greatly simplifying team onboarding and collaboration. Indeed, the vision the Che team has held for years is to allow anyone to contribute to a software project without the need of installing additional software[8]. Compatible with multiple container orchestration systems, Che supports both single-user as well as multi-user development, which makes it more adaptable and supportive for teams. Furthermore, integrated within Red Hat's OpenShift.io platform, Che enables developers to run it as SaaS.

In addition, Eclipse Che is designed to be a platform where developers can also build and integrate their own development tools. By writing and adopting extensions, it is up to the vision and creativity of Che's users to create new solutions. This way, Che offers a similar to its desktop Eclipse counterpart - developers are given an extensible development environment, but accessible through the cloud[9].

In this report we analyze and provide a high-level understanding of the architecture of the Eclipse Che project. We first identify the key stakeholders of the project and describe the context of Che. We then analyze the technical debt and offer more insights from two viewpoints: a development view and a functional view. Since the Eclipse Che team just released both 6.19.3 and 7.0.0-beta-3.0 versions[10], we consider both Che 6 and 7 and discuss what the new Che 7 brings to the developers.

4.1.1   Stakeholders

We have identified stakeholders of Che by analyzing the Che repository[11], Github's Insights and the official Eclipse Che website. We will formulate this list mainly following the scheme defined by Rozanski and

---

[1]Benjamin Mestrallet. From eXo Cloud IDE to Codenvy Raising 9 Million Dollars: A Brief History, 2013. exoplatform.com

[2]Benjamin Mestrallet. From eXo Cloud IDE to Codenvy Raising 9 Million Dollars: A Brief History, 2013. exoplatform.com

[3]https://www.eclipse.org/org/foundation/]

[4]https://www.eclipse.org/ecd/

[5]Tyler Jewell, Introducing Eclipse Che and Eclipse Cloud Development, 2014. codenvy.com

[6]Tyler Jewell, Red Hat To Acquire Codenvy, 2017. codenvy.com

[7]Tyler Jewell, Introducing Eclipse Che and Eclipse Cloud Development, 2014. codenvy.com

[8]Tyler Jewell, OpenShift.io and Eclipse Che, 2017. eclipse.org

[9]Tyler Jewell, Introducing Eclipse Che and Eclipse Cloud Development, 2014. codenvy.com

[10]Eclipse Che: Release}, 2019. https://github.com/eclipse/che/releases

[11]Che Wiki: Development Process, 2019. https://github.com/eclipse/che/wiki/Development-Process

Woods in the book *Software Systems and Architecture*[12],[13].

4.1.1.1 Acquirers

Codenvy donated the Che kernel to Eclipse Foundation and has since been acquired by Red Hat, all while still continuing to build its commercial product on top of Eclipse Che. We thus consider Eclipse Foundation and especially Red Hat as the acquirers of Che.

4.1.1.2 Assessors

The project has two Eclipse Foundation assigned mentors - Mik Kersten and Marcel Bruch, who is also a member of the Eclipse Foundation *Architecture council*. They are the assessors of Che, as their main concerns are Che's continued technological success and innovation, widespread adoption, and future growth [14].

4.1.2 Communicators

Everyone with a role in Che's Development Process[15] - including contributors, committers, reviewers, triagers, maintainers and project leaders - is expected to be a communicator of Che, since they understand the details of the architecture and can explain them, for example, to technical authors responsible for the documentation.

4.1.3 Developers

All the contributors and committers to the project on GitHub are developers of Che. Most of the contributions come from Codenvy / Red Hat employees since Codenvy built the kernel of Che and the two organisations remain the major stakeholders of the project.

We list the top-five contributors by the number of commits (as of April 2019):

- riuvshin
- sleshchenko
- garagatyi
- benoitf
- mshaposhnik

---

[12]Nick Rozanski and Eoin Woods. Software systems architecture: working with stakeholders using viewpoints and perspectives. Addison-Wesley, 2011.

[13]Software Systems Architecture, 2019, https://www.viewpoints-and-perspectives.info

[14]Che Wiki: Development Process, 2019. https://github.com/eclipse/che/wiki/Development-Process

[15]Che Wiki: Development Process, 2019. https://github.com/eclipse/che/wiki/Development-Process

4.1.4   Maintainers

The Che development team lists all the project maintainers[16], assigning them to different product areas including IDE, language servers, platform, dashboard, QA automation, and others.

The maintainers are responsible for managing the issue backlog, ensuring code quality and guiding technical decisions[17]. Maintainers also periodically review pull requests (PRs) related to their respective product area - this is to make sure the PR reviewer lists are correct as well as to oversee the quality of the reviews themselves.

4.1.5   Integrators

The integrator tasks are mostly done by the maintainers and codeowners. PR merges require the approval of at least one of the maintainers[18][19]. PRs that include documentation and release notes updates must also include a project manager (PM) reviewer. In addition, Tyler Jewell, one of the project leaders, can authorize the merging for any PR. The integrators take care that the reviewers focus on code style, readability of the code and usability for other developers.

Contributors with commit rights will merge their own PRs after all reviews are done, while those without commit rights will have the maintainer merge the PR instead. In addition, maintainers are responsible for periodically reviewing the open PRs[20].

4.1.6   Suppliers

Che is implemented mostly by Java and runs on top of Apache Tomcat by default. Part of the project is developed using TypeScript. It is built upon Docker 1.8+ and Maven 3.3.1+. The IDE used inside the browser used to be written using GWT until Che 6 and has been using Theia since Che 7.

The Che project uses GitHub for code version control, issue tracking and releases, and Jenkins CI server hosted by Codenvy for continuous integration.

4.1.7   Support Staff

Anyone can ask for support and request features using GitHub issues. Weekly community meetings[21], a public Mattermost channel and a developer mailing list are all open for everyone to join and ask for help. Dave Neary, a Red Hat open source community manager, also acts as a community manager for Eclipse Che.

---

[16]Che Wiki: Development Process, 2019. https://github.com/eclipse/che/wiki/Development-Process

[17]Che Wiki: Development Process, 2019. https://github.com/eclipse/che/wiki/Development-Process

[18]Che Wiki: Development Process, 2019. https://github.com/eclipse/che/wiki/Development-Process

[19]Che Wiki: Development Workflow}, 2019, https://github.com/eclipse/che/wiki/Development-Workflow

[20]Che Wiki: Development Process, 2019. https://github.com/eclipse/che/wiki/Development-Process

[21]Che Wiki: Community Meetings, 2019, https://github.com/eclipse/che/wiki/Che-Dev-Meetings

4.1.8 System Administrators

In a company or organization, Che may be deployed by IT specialists in which case the IT department takes on the system administrator role. System administrators are concerned with Che's deployment, updates, plugin installations, and should be the first point of contact for users if anything goes wrong with Che.

Individual users who download, deploy and develop projects with Che are considered system administrators themselves. When Che is used as SaaS hosted at che.openshift.io, OpenShift assumes the role of system administrator instead.

4.1.9 Testers

All contributors are supposed to be testers and are responsible for the code they submit. The Che development team requires all contributors to follow the Development Workflow[22] for the purpose of submitting high-quality code.

4.1.10 Users

We identified three classes of Che users.

Application developers are the primary users of Che. They can use it either with via the embedded browser IDE or instead with a different desktop IDE by mounting the Che workspace filesystem via SSH.

Product owners use Che by managing settings on the workspace server to provide on-demand workspaces for developer teams.

Extension providers make use of the Che SDK to extend and provide new customizations for Che.

4.1.11 Competitors

There are several competitors to Che on the market, some of them open sourced and some not. Here we mention three main competitors:

- Cloud9 IDE is another online integrated development environment and supports multiple programming languages developed by Amazon. It is written almost entirely in Javascript and published as open source from version 2.0.
- Azure DevOps, formerly called VisualStudio Online before it was rebranded as Visual Studio Team Services (VSTS), is an online platform that supports Team Foundation Server and rolling release models.
- Codeanywhere is another cross-platform cloud IDE created by Codeanywhere, Inc. It is written entirely in Javascript.

---

[22]Che Wiki: Development Workflow}, 2019, https://github.com/eclipse/che/wiki/Development-Workflow

4.1.12   People to Contact

We sent an email to the mailing list and showed our interest in talking with some developers. Dave Neary, the community manager, replied and invited us to attend the Che weekly meeting on April 1st, 2019 in which three developers Thomas Maeder, Stevan LeMeur and Sun Tan shared their opinions with us about why they thought Che is amazing and unique and challenges they've found, which we recorded in a video.

4.1.13   Power-Interest

In the figure below we classify stakeholders of the Che project using the power-interest grid[23]. It indicates different levels of interest and power of stakeholders on the project. We then categorize the stakeholders into four groups in terms of how closely they need to be managed: key stakeholders which are to be managed closely, stakeholders to keep satisfied, ones that require minimal effort, and ones to keep informed.
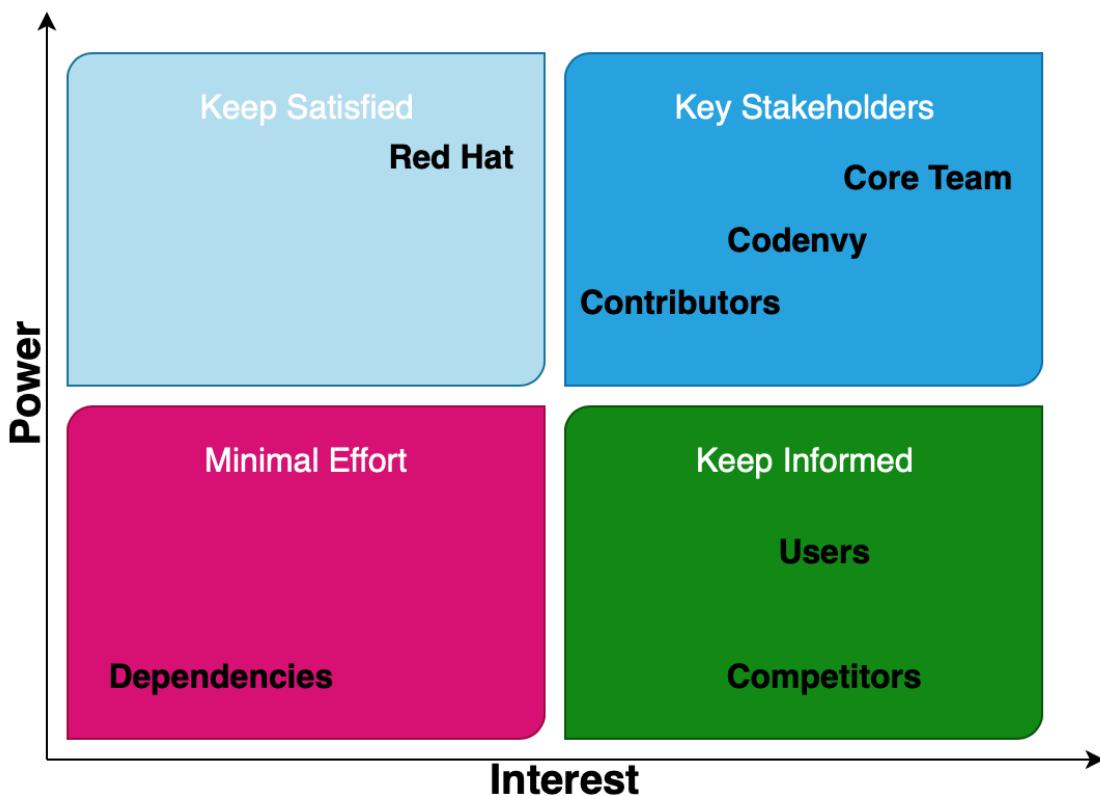
Figure: Power vs. interest grid for the Eclipse Che project.

---

[23]Aburey Mendelow, Stakeholder mapping. Proceedings of the 2nd international conference on information systems, Cambridge, MA, 1991.

4.2 Context View

In this section we give a further overview of the scope of Eclipse Che and describe how it is positioned within its context.

4.2.1 System Scope

Eclipse Che in its current form offers[24]:

- containerised development *workspaces* with an integrated browser IDE and runtimes configurable to include all project runtime dependencies the developers need
- *a workspace server* that orchestrates the workspace containers and deals with user management
- *plugins* which add support for various development services, including IDE support for specific programming languages, frameworks, and developer tools like debuggers
- an *SDK* for creating new plugins and shipping them by building new Che assemblies

4.2.2 Context Model

We expand on the details already discussed in previous chapters by looking at the entities the Eclipse Che project interacts with and categorising them, presenting them in a diagram, and briefly describing each of the newly introduced ones.

4.2.2.1 Organisation

With Codenvy having been an important driving force of the project since the very beginning, the list of the current project leaders is not surprising - Tyler Jewell is the Codenvy founder and former CEO, Gennady Azarenkov was the Codenvy CTO and now works at Red Hat, and Brad Micklea moved from Codenvy to RedHat as a product manager. Furthermore, the project has two Eclipse Foundation assigned mentors.

4.2.2.2 Development

Eclipse Che is a *Java* project which uses *Maven* with several testing plugins as the project builder and package manager. New releases of Che get built by a *Jenkins* CI server, while *Sonarcloud* is used for continuous code quality checking and *Swagger.io* for building and documenting APIs.

4.2.2.2.1 Connected services Che runs on top of an application server and is deployed with *Tomcat* and *PostgreSQL* out of the box. To build its IDE, Che relies either on *Eclipse Orion* (Che 6) or *Eclipse Theia* (Che 7).

Container orchestration is essential to install, run and manage Che, and the orchestrators currently supported are *Docker*, *Kubernetes* and *OpenShift*. For multi-user Che, *Keycloak* is used to handle user authentication and access management.

---

[24]https://github.com/eclipse/che-docs]
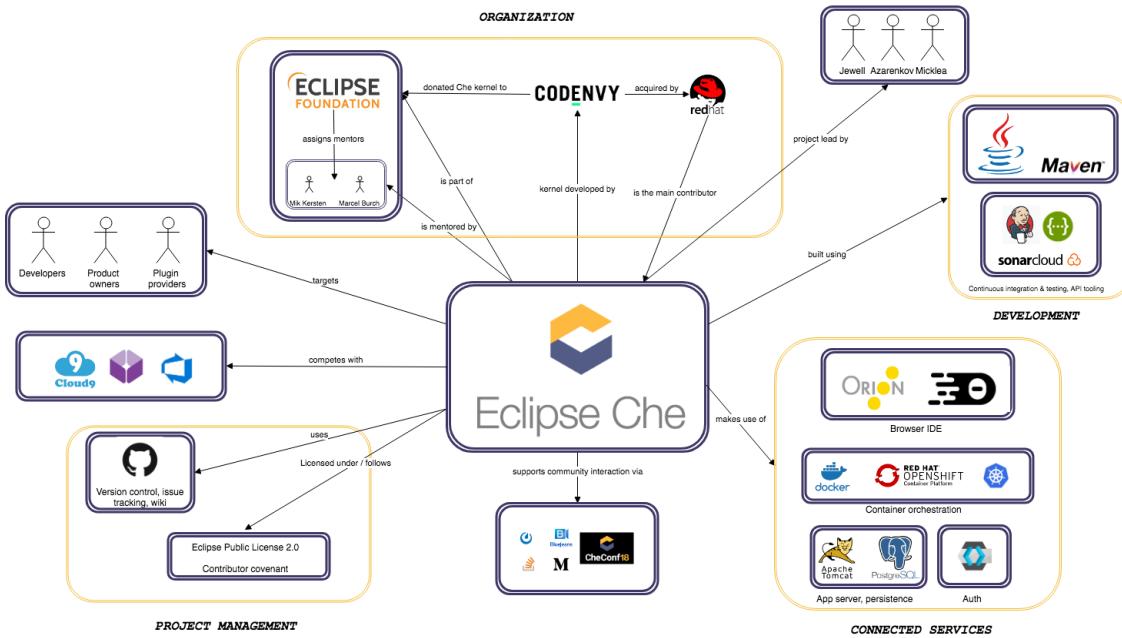
Figure 4.2: context model diagram

4.2.2.2.2   Project management   Most of the project management happens on *GitHub*, including issue tracking and version control. The project also maintains a GitHub Wiki that details the project management and development processes.

4.2.2.2.3   Licensed under / Follows   Eclipse Che is licensed under the *Eclipse Public License 2.0* and its maintainers follow and enforce the *Contributor Covenant* Code of Conduct.

4.2.2.3   Community and support

The Eclipse Che community is active on multiple communication platforms. The project has an active *mailing list*, a *Mattermost* channel, maintains a blog presence on *Medium*, and organises biweekly videoconference Dev Meetings via *BlueJeans*. The project also organises an annual virtual user conference *CheConf*.

4.3   Technical Debt

4.3.1   Technical debt in Eclipse Che

The project is split up in several modules. This helps in keeping coupling low, because different modules can only interact if there is an explicit dependency defined in maven. Most classes are very small, which helps in keeping cohesion high.

Running Sonarcloud on the source code of Che some interesting outliers are found. One of these is `ClientTestingMessage` in the testing plugin. This class violates SOLID by keeping a reference to all classes that inherit this class. This can also cause further problems.

Another class which grabs the attention is `JGitConnection`, which is by far the largest class in the project. This class is a very large adapter class for jgit, it implements everything needed for git interaction in a single class.

The `CronExpression` class has the largest amount of technical debt according to Sonarcloud. This class is very large, where it probably should not be. There are methods with very high complexity, there is duplicated code and possible null pointers. Given that this class implements something very specific that has nothing to do with this project, it should probably be moved to a dependency.

The `MenuLockLayer` and `TextBox` classes have a D and E rating respectively. Both these classes extend a class from GWT. The main issue with these classes according to Sonarcloud is that the inheritance tree is too deep (larger than 5). It wouldn't be viable to fix these issues, because the root of the issue lies in the dependency. For the next version of Che, these classes are deprecated because a new IDE is introduced.
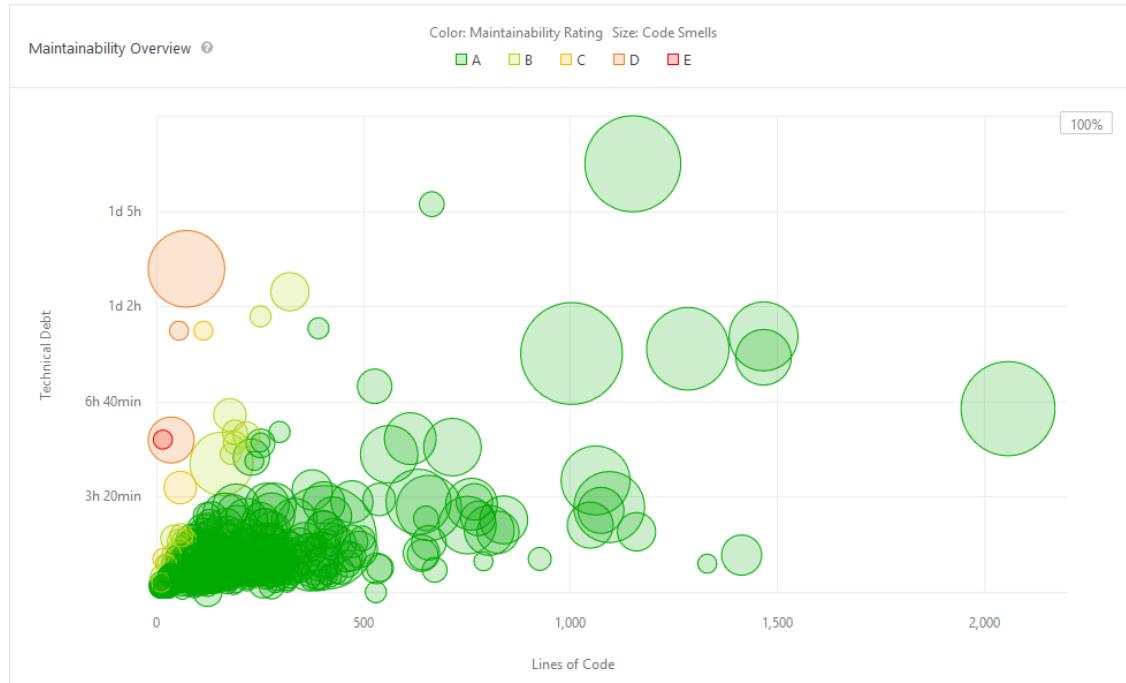
Figure 4.3: Sonar maintainability rating

error prone is used to report warnings about common errors.

4.3.2 Test debt in Eclipse Che

Eclipse Che is tested with JUnit (4900 Unit tests for individual modules) and Selenium (640 Integration tests between modules). The Selenium end-to-end tests test Che in a browser with an automated user. Building

and running the tests takes about three hours. The end-to-end tests take about 2 hours to complete. These tests are only executed in pull requests when it is needed. There are also nightly builds which allows QA to test changes rapidly. During development the contributor SkorikSergey focusses on proper testing and gives advice on this for other contributors in Pull Requests.

The test reports are run in a Jenkins Continuous Integration (CI) environment. In addition, a SonarCloud environment is available, but hasn't been used since November 2018. Code coverage isn't taken directly into account into the Jenkins CI or SonarCloud environment.

We performed an analysis on the tested modules and the reports generated by a recent SonarCloud analysis by our team. In the testing degree Figure below, the lines of code are plotted against the number of tests written for that module. The size of the bubble indicates the relative number of code smells reported by SonarCloud.

In general, for the tested modules it is found that modules with more lines of code are tested with more tests as well. Modules which are tested with relatively more tests tend to show a lesser degree of code smells (docker-client, infrastructure-kubernetes) than modules with a lower number of tests (`che-core-api-core`, `che-core-commons-gwt`).
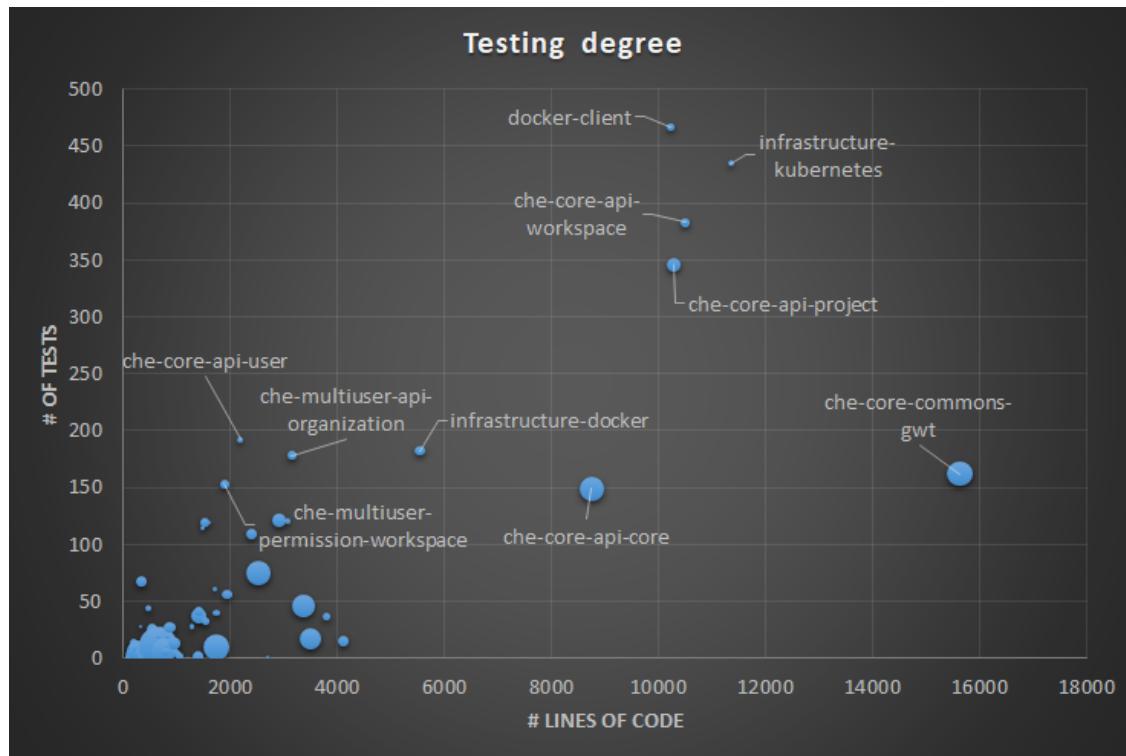
Figure 4.4: Overview of the tested classes, the size of the bubble represents average Code Smells per line of code.

In addition, it is observed that there exist quite some tests for the modules responsible for permissions. This makes sense as assurance of the access security is very important to gain trust from users. Bugs which lead

to unauthorized access often lead to higher damage than functional bugs. On the other side, plugin modules are barely or not tested at all. The choice for less testing might lie in the more trivial code involved.

Our Sonarcloud analysis showed that in some modules there exists some code duplication. For example, in the che-core-api-dto module 21.1% duplication is observed. However, additional analysis indicated that this module is generating java code and deduplication in this module will likely lead to less maintainable code. Lastly, it is found that for the Selenium tests, additional tests exist which test whether the Selenium tests behave accordingly. These additional tests test whether mocked classes have the right intended behaviour for example.

In general, testing is performed well in the project. Actions which can improve the testing approach is to enable automatic coverage tools. This provides an indication of what is and isn't tested in the project. Furthermore, it can be smart to make more use of the SonarCloud environment.

4.3.3 Discussions about technical debt

There are sprint issues in the repository which are created for every two weeks. These issues have a specific section on Technical Debt. In this section, there is a list of issues which are considered technical debt and that should be fixed in that sprint. Issues themselves are not marked as containing technical debt.

There are about 150 TODO comments in the source code. Most of these were added in 2017 and there are two authors who have contributed 90% of the TODO comments. It seems that most of these TODO comments don't require some implementation as they have already been in the code for almost two years. Several comments are in the GWT IDE, which is no longer under active development and is a candidate for being moved to another repository.

4.3.4 Evolution of technical debt

The Eclipse Che team has made a recent move from GWT to Theia as IDE with the release of Che 7. Theia is maintained in a separate repository and can also be used outside of Che. This movement makes a coupling with Che very low and better maintainable. The team has moved to the VSCode plugin protocol as well. This enables better modularisation for plugins. Both movements help in lower technical dept.

Not much historical data is easily available as SonarCloud isn't run very often and older CI builds aren't preserved long. It is found that tests are continuously developed during development. Additionally, it is found that during development the number of code smells is also highly reduced.

To get some more insight into the evolution of the project, we ran several major releases in our sonarcloud environment 4.0.0, 5.0.0, 6.0.0, 7.0.0-beta-1.0.

It was observed that the number of code smells highly dropped from Che version 5 onwards. Between version 6 and 7, a major drop in code smells can be attributed to the removal of forked code from Eclipse JDT in version 6.13.0. In addition, other dependencies are removed as well due to the move from GWT to Theia.
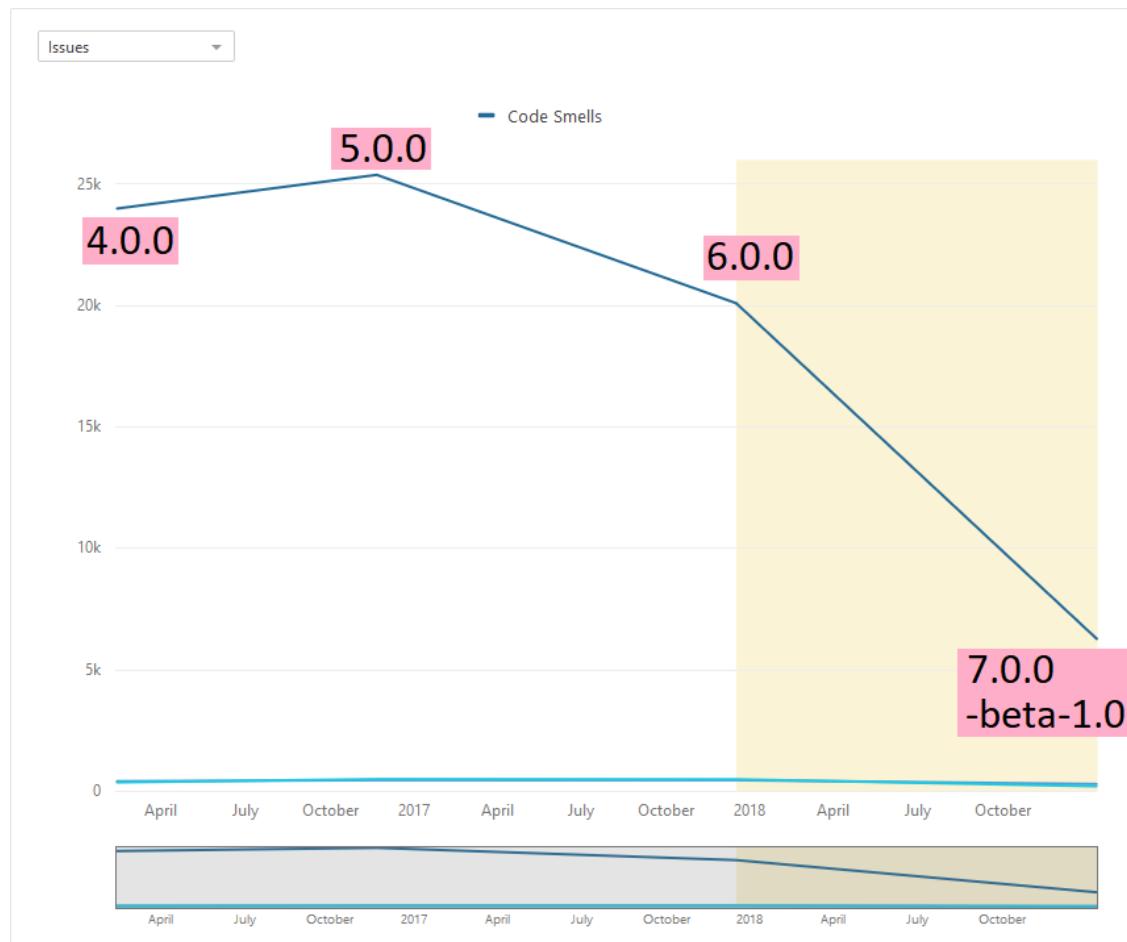
Figure 4.5: Evolution of number of code smells in Eclipse Che

4.4 Development view

4.4.1 Module structure

Eclipse Che source code is modularised and uses Maven to manage the dependencies between the modules. On a high level, we analyse the module dependencies by looking at the way the Che assembly binaries of the main system components are built. The components we considered are thus: the Che workspace master (that is, the Che server), workspace loader, workspace agents (that run and manage a workspace instance), IDE, and User Dashboard.

From a top level (in the Figure below), both the workspace server and the agents make use of the Che Core module which contains shared APIs and libraries, while the Dashboard is a self-contained Web-based module. The workspace agents make extensive use of Plugin modules which support IDE and other workspace services. We follow the dependencies of both the workspace server and the agents further in Figures below.
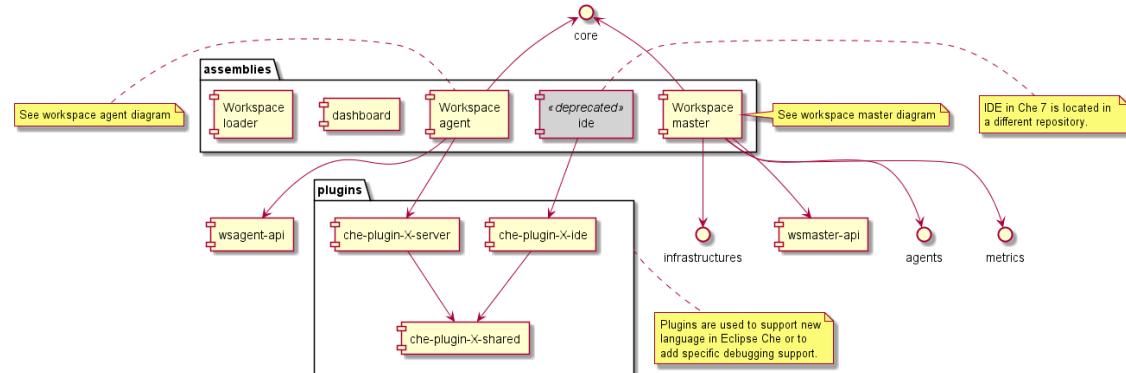
Figure 4.6: Top level module relationships
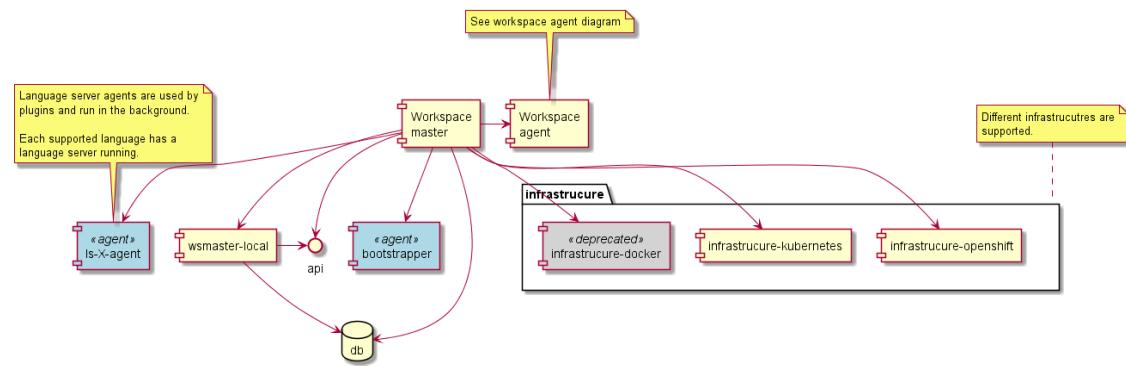
Figure 4.7: Workspace server module relationships
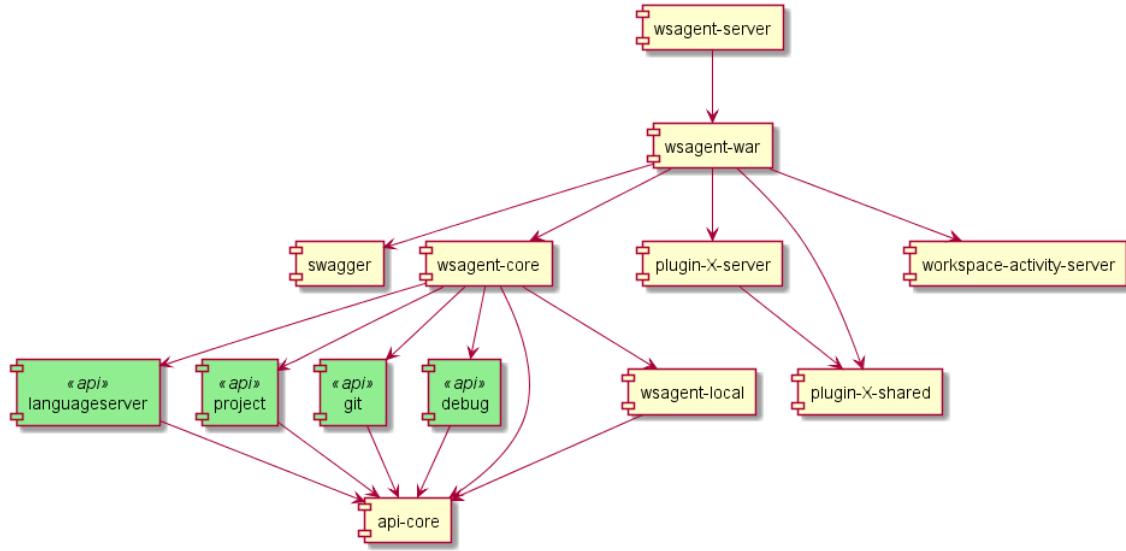
Figure 4.8: Workspace agent module relationships}

4.4.2 Common design model

4.4.2.1 Core APIs

The Core APIs used in Eclipse Che are in the `core/che-core-api-core` module, including definitions of JSON-RPC and REST operations, event subscription, unsubscription and notification, web socket, proxy authenticators and password encryptors, and standard API exception formats.

4.4.2.2 Database interaction

Database interactions are defined in the `core/che-core-db` module — for example, database initialization and termination, SQL script creation and migration, common database error codes as well as data source tracing methods.

4.4.2.3 Shared libraries

Most of the shared libraries for the server, agents and plugins are in the `core/commons` module. Some examples of these libraries include ones that handle Single Sign-On (SSO) token extraction and authentication exceptions, JSON parsing and writing, and e-mail services.

4.4.2.4 Logging and instrumentation

Che uses SLF4J and Logback for logging within the browser IDE and on server-side, respectively. Che also exposes metrics in a format that can be consumed by a Prometheus server and visualised in Grafana - this

includes metrics from the JVM, the class loader, and the Tomcat server. Che also supports the collection of end-to-end trace data by deploying a Jaeger server.

4.4.2.5 Third-party libraries

Eclipse Che relies on a set of third-party libraries each of which serves independently, such as Google Web Toolkit and Tomcat, SLF4J for message logging, Swagger for API design, and Javax for email services and web socket applications.

4.4.2.6 Design pattern: "Sidecar container"

In this cloud development design pattern, the sidecar containers are attached to the parent application container and are used to provide supporting features to the main application. Sidecar containers can follow their own development and runtime lifecycle, allowing for easy upgrades and modifications.

This design pattern became prominent with the new Che 7. Previously, all the dependencies required for active development within a workspace had to be injected into the main workspace container image. This meant that the development container image used in the workspace was not identical to the production image anymore. To address this and avoid having to modify the production image clone, the sidecar container pattern was employed in order to organise the workspace containerisation.

The various services that used to be parts of a workspace have thus been split into separate containers (as seen in the Figure below), providing further isolation and encapsulation. The main workspace container does not need to have extra plugins installed anymore, as any additional workspace services and plugins will run in separate sidecars.

4.4.2.6.1 Plugin brokers Plugin brokers are special services that will, given a specific plugin metadata description, gather all the information about definitions already available on the Che server to and prepare the workspace for the plugin installation. This includes, for example, downloading and unpacking required files, returning a set of endpoints, containers (and even editors) for the plugin. The main goal of using a plugin broker is to decouple the Che plugin setup from the Che server itself, allowing Che to support different kinds of plugins without having to modify the Che server code for every new kind of plugin added.

4.4.3 Codeline model

4.4.3.1 Source code structure

We describe the overall, high-level structure of the source code repositories used by the project. The main codebase of course resides in the main Che GitHub repository; however, some of the source code the project depends on is contained in other repositories in order to allow for it to be managed separately.

4.4.3.1.1 Main Repo The main Che source repository on GitHub has the following high-level structure[25], which generally follows the way Che is split into submodules.

---

[25]Che Wiki: Development Workflow}, 2019, https://github.com/eclipse/che/wiki/Development-Workflow
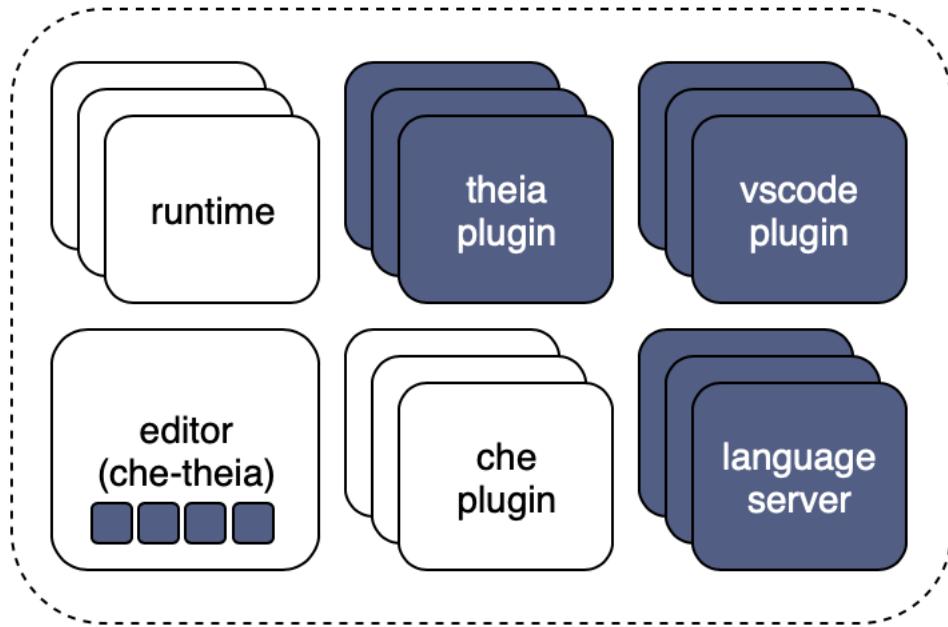
Figure 4.9: Sidecar containers

Directory	Description
/	Root directory
/agents	Workspace software (web terminal, language servers, ssh server)
/assembly	Generates binary assemblies of Che
/core	Shared libraries for server, agents, and plugins
/dashboard	JavaScript app user management
/dockerfiles	Docker images to run Che, CLI, & utilities
/ide	Browser IDE
/plugins	IDE & workspace agent plugins
/samples	Code templates for fresh workspaces

4.4.3.1.2 Other Repos   Some of the Che dependencies are managed in other repositories owned either by the Eclipse Foundation or Codenvy. This is done for multiple reasons[26] — these dependencies are either forks of other relevant projects, contain libraries that follow a different tagging lifecycle than Che, or they contain very large files.

Repository	Description
eclipse/che-dev	Dev resources - code style, license headers
eclipse/che-dockerfiles	All project's Dockerfiles
eclipse/che-docs	Official Che docs
eclipse/che-lib	Dependencies forked from elsewhere (antlr, Orion, Tomcat, Logback, Swagger...)
eclipse/che-parent	Maven root parent POM, manages dependencies
eclipse/che-theia	New Che 7 IDE
codenvy/che-installer	Windows and JAR installers
codenvy/che-tutorials	SDK examples and tutorials
che-incubator/chectl	Command line interface for Che
redhat-developer/rh-che	Eclipse Che on OpenShift
redhat-developer/devfile	Che 7 devfile specification

The source code for the official Eclipse Che website is tracked on the Eclipse Foundation git repository:

4.4.3.2.1 Building Che Che manages dependencies and builds the project with Apache Maven. Docker is used to provide container images that contain Che and all its dependencies. In addition, the Web-based user dashboard also makes use of [yarn](https://yarnpkg.com/en/], webpack and gulp.js for package management and building, respectively.

4.4.3.2.2 Continuous integration Continuous integration of committed code changes is executed on a Jenkins server hosted by Codeenvy. The Eclipse Che project runs several Jenkins CI jobs which are either "plain" builds (CI builds on git push to master, PR builds triggered via PR comments, nightly builds) or specialised quality assurance builds with further testing (ran nightly, for a release candidate, or for the final release version).

4.4.3.2.3 Testing Below we give an overview of the kinds of automated testing done when working on Che.

Test / analysis type	Used by Che
Unit testing	JUnit, TestNG at build time
Integration testing	JUnit, TestNG at build time
Functional testing	Selenium
Static analysis	FindBugs, Error Prone at build time
Code style	Maven formatter plugin
Continuous code quality	Sonarcloud

Unit and integration tests are run at build time using the Surefire plugin for Maven. Static analysis is also integrated into the build phase with both FindBugs and Error Prone checking for compile-time detectable Java bugs. Code style is checked using a Maven plugin that checks for compliance with *Google Java style* formatting.

Further quality assurance is done via functional and end-to-end testing using Selenium.

Sonarcloud has been used for continuous static analysis of Che releases between versions 6.2 and 6.14 (the last check was done in November 2018). It is unclear why continuous code quality checks do not seem to be run on a regular basis anymore.

4.4.3.3 Release process

Successfully built and tested new versions of Eclipse Che are released at the end of each 2-3 week development cycle ("sprint"). The release process itself is automated and integrated into the standard CI pipeline. Che *binaries* are tagged and compiled from release version source code using Maven, whereas Docker is used to build and tag *container images*. If blocker bugs are found during the final quality assurance phase, the release can be blocked and postponed until the end of the next sprint.

4.4.3.4   Configuration management

Software configuration management prescribes how changes are tracked and managed in order to ensure that repeatability and traceability are possible when collaborating on a software project. Here we detail the tooling and configuration structures employed by the Che team to support the development lifecycle.

4.4.3.4.1   Tooling   We first specify the configuration management tooling used by the Che team. We categorise them following Berczuk's list of tools needed for configuration management.

Tool type	Used by the Che team
Version control	Git
Build Tool	Maven
CI Server	Jenkins
Artefact Repo	Nexus
Doc System	GitHub Wiki

4.4.3.4.2   GitHub Configuration Structures   The Che team uses GitHub to track and control the development of lifecycle. We identify the type of codelines used and their respective GitHub branches.

There are three types of codelines used:

- **Mainline**: The master branch containing the latest code which is well-tested and stable (as reported by the CI system),
- **Active development lines**: Multiple branches where new features or bug fixes are worked on (are "in progress" and are allowed to be failing)
- **Release line**: One branch at a time which contains the source code of the latest Che release.

The active development lines can live either in the main repository if the authors have contributor access or in project forks. The latter is discouraged, however, when collaborating on long-lived feature branches - it is easier to track changes if all the development happens on main repo branches. In addition, feature branches have to be named after the matching GitHub issue number and must be removed after they are merged to mainline.

When preparing to release a new version of the software, a release branch is created from the master branch on the release candidate commit. Any potential bugs found after the release are fixed on this release branch and the branch itself is reused if a bugfix release is necessary. Once a new release is in progress, the previous release branch gets removed.

The Che team does not make extensive use of *tagging*. The only tags used are release version numbers labelling the respective commits where the past and the current release lines branched off.

4.5 Functional View

Eclipse Che consists of several components which together form the development environment[27]. The functions of these components in the architecture, together with their interactions are depicted in the functional view. Projects are contained in workspaces which are hosted in a workspace server. These workspaces can subsequently be accessed by browser-based IDEs. In addition, plugins can be implemented for which a Software Development Kit (SDK) is available.
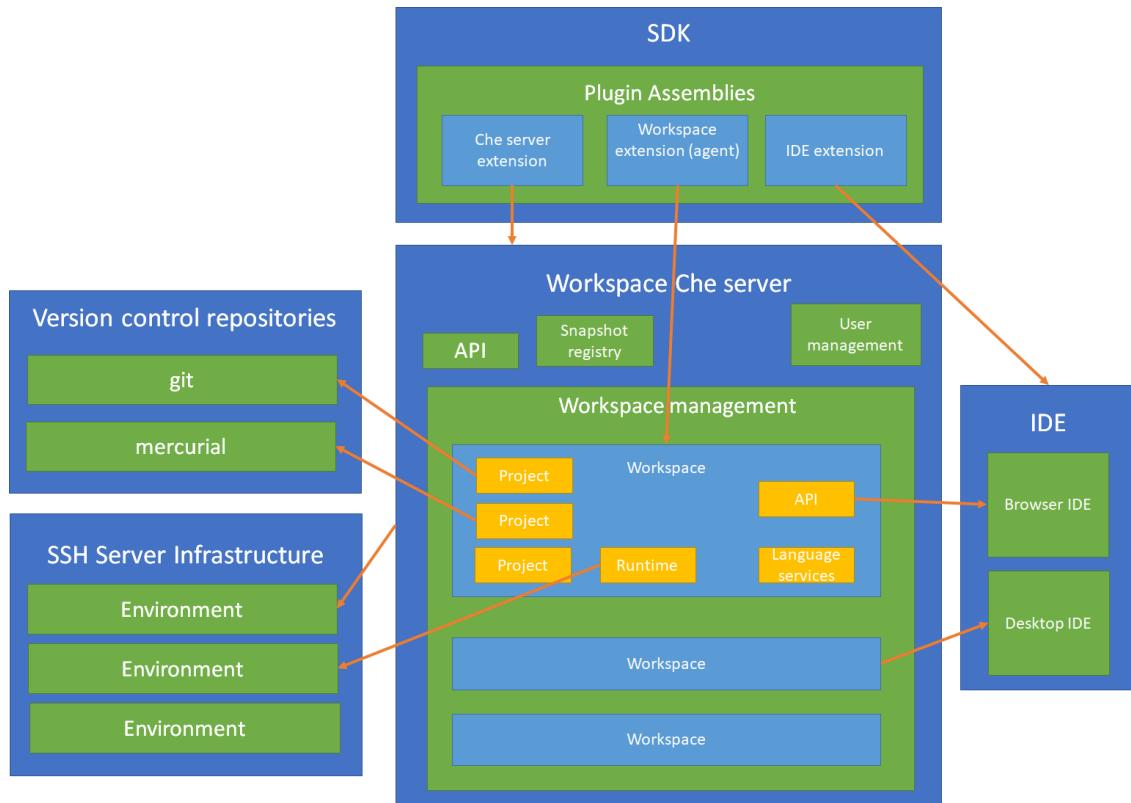
Figure 4.10: Functional architecture

4.5.1 Workspace server

The workspace server of Eclipse Che is installed on a Server infrastructure (for example Apache TomCat) and is the central access point for the workspaces it maintains. Single or multiple users can access workspace via a Restful API connection. The different workspaces are isolated from each other and cannot interact.

---

[27]https://github.com/eclipse/che-docs]

4.5.1.1 Workspaces

Workspaces are used by one or multiple users to develop projects. A workspace can have any number of projects which can be connected to version control repositories (like git, mercurial or subversion) as well. Every workspace can be accessed through an IDE specific for that workspace via the browser. An API is used for interaction between the browser IDE and the workspace. In addition, access via desktop IDEs is possible as well. Language services enable development support for different languages of choice. Every workspace has one or multiple runtimes, using a host environment. These runtimes are used to build, run, and debug projects like one would do locally with for example maven. Workspaces can be exported to move them from one Che server to another. In addition, it is possible to save and restore snapshots via a snapshot registry.

4.5.2   Server infrastructure

Server infrastructures are used to make environments for the Che Server as well as environments for the workspace runtimes to run their code in. The server environments make use of either Docker or OpenShift to work. Users can either set up these on their local PC or on a central server. Local installations can be used by single users for small projects. Such an installation is easier to start with as no login is required[28]. Server installations can be used by multiple users in development teams in organizations. In addition, it is also possible to use a server from Redhat in the form of Software as a Service (SaaS).

4.5.3   SDK

Eclipse Che is made so it can be extended easily. Plugins can be made by plugin providers for either the Che server, a workspace (also called an agent) or the browser IDE. For the development, a Software Development Kit (SDK) is available which enables the packaging plugins into assemblies for use in production. Several plugins are already available for elements like ones for languages, frameworks and tools.

4.6   Conclusions

We have analyzed the architecture of Eclipse Che, an open-source cloud IDE. It offers containerized development workspaces and plugins to support various development services. Codenvy, Red Hat and Eclipse Foundation are considered the most important stakeholders. We have identified integrators and pull request merge strategies by pull request analysis. Each pull request needs to pass the tests and code reviews before a maintainer approves its merge. The technical debt analysis shows that the team is working hard to remove components from project which have a large amount of technical debt. We also observed that tests help properly in reducing code smells.

The development view provided us with more insights into the project. We have analyzed the module structure, the codeline model and the common design model. Here the sidecar is one of the most interesting design patterns used. The functional view depicts the components of Eclipse Che and their interactions.

---

[28]Single and Multi-User Che, 2019. https://www.eclipse.org/che/docs/che-6/single-multi-user.html

4.7 References

Chapter 5

Cockpit

5.1   Table of Contents

5.2   Introduction

The Cockpit-project is an open-source project owned by Red Hat, that provides an easy-to-use, integrated, glanceable, and open web-based interface for interacting with the underlying servers. More specifically, it allows users to monitor their system and adjust server configuration on GNU/Linux-based server operating systems such as RHEL, Debian, Fedora and many others. Apart from monitoring and interacting with a local server, Cockpit can also be used to access multiple remote servers/clusters and interact with them from the same user interface.

Although Cockpit is an open-source project, it consists of a small community of active contributors, mainly Red Hat employees. The primary reason for this is the complexity of the project and due to the fact that the core team is not that open to external contributors.

The following sections include an in-depth analysis of Cockpit's architecture. More specifically, we first analyze the stakeholders and the merging pipeline based on existing pull requests. After that, we examine Cockpit's architecture from three different perspectives, namely the Context View, the Deployment View and the Development View. In the sections that follow, we attempt to analyze the project's Technical and Testing debt and try to identify the evolution of the project throughout its release-history. Lastly, we summarize our findings and present our conclusions regarding the project's architecture.

5.3 Stakeholders

5.3.1 Acquirers

Red Hat (which was recently acquired by IBM) is undoubtedly behind the funding of the Cockpit-project. They employ the core team which is responsible for the evolution of the project and oversee the progress of the project's development.

5.3.2 Assessors

There is no clear identification of assessors that specifically focus on the system's conformance. It is most likely that the core team ensures the conformance to the standards and legal regulations. The Cockpit-project uses the LGPL v2.1+[1] license which means that everyone is allowed to copy, distribute and modify the project as long as the modifications are described and licensed for free under LGPL. However, although not mentioned directly, there must be some form of legal counselling for cases where violations are observed.

5.3.3 Communicators

In the Cockpit-project this category consists of the core team itself.

Other types of stakeholders can learn more about the project via the team's irc channel, mailing list, or via the project's blog. The authors of the blog and their published posts are listed here. It should be noted though, that different members have different roles as communicators. For instance, Stef Walter([@stefwalter](https://github.com/stefwalter)) has presented the project in conferences such as the Devconf, whereas others are responsible for updating the blog with new releases or creating usage and contribution manuals for the system.

5.3.4 Developers

The developers of the project are practically every individual willing to spend time and effort in understanding the project's architecture and standards. This category includes anyone that actively participates in the discussions and contributes to the project's progress. The top 3 contributors were found to be Stef Walter ([@stefwalter](https://github.com/stefwalter)), Marius Vollmer ([@mvollmer](https://github.com/mvollmer)) and Martin Pitt ([@martinpitt](https://github.com/martinpitt)) with 4.164, 1.769 and 1.435 commits respectively.

5.3.5 Maintainers

The project is maintained by the community, adhering to a set of guidelines. These can either be found in the project's website (code style, commit workflow, and interface design) or in the github repository. In general, changes can be proposed by pull requests.

The documentation is also hosted on the Cockpit website.

On the Cockpit wiki the maintenance is listed explicitly. Both Stef and Martin are named as responsible.

5.3.6 Suppliers

The end users are responsible for setting up the hardware and infrastructure of their running system and as a result can be considered as suppliers. Moreover, the system is designed to work on a variety of different browsers (e.g. Chrome, Mozilla) and Linux distributions (e.g. RHEL, Fedora, Ubuntu) each of them requiring different versions. The exact specifics can be found at the installation guide. Since these dependencies impose limitations they can definitely be categorized as suppliers.

5.3.7 Support Staff

The core Red Hat team along with the contributors that have sufficient knowledge of the system are the ones who are able to support the users of the system. This can either be done via the irc channel of the project or by joining the mailing list. Although there is no contact page, there is documentation on both the project's and Red Hat's website.

5.3.8 System Administrators

Every end-user of the project is responsible for handling the system's administration from their end.

5.3.9 Testers

There is not a particular individual or team (e.g. QA testers), that mainly takes on the task of conducting tests. Every contributor is responsible for writing tests about the part that was implemented and when a pull request is issued, experienced core team members review and test the implemented functionality. However, testing is just another task for them while reviewing and by no means they can be called pure testers. Finally, a side project (Cockpituous) is also utilized for the CI/CD pipeline of the main project. This is how the automated unit and integration tests are being run.

5.3.10 Users

The Cockpit-project is useful for every developer that is interested in monitoring the underlying web server(s) of their system via a web interface. However, the project is mostly suitable for system administrators (devops) as it offers an easy way to monitor and interact with the servers without purely relying on a command line interface for the related operations.

5.3.11 Other stakeholders

Although there are not many other stakeholders that can be found on the GitHub page and official website, we can still identify a few:

5.3.11.1   Integrated platforms

Cockpit is integrated in the repositories of a few Linux distributions by default. These distributions are: *
Fedora * RedHat Enterprise Linux * Project Atomic * CentOS * Debian * Ubuntu * Clear Linux

These distributions can also be considered stakeholders, since they benefit from Cockpit being available to
their users via their default repositories, making installation on their platforms easy.

5.3.11.2   Competitors

Cockpit has three main competitors: Ajenti, Webmin and Nagios, which are all competing server manage-
ment software packages. All of these competitors are also open source projects, with Nagios having both a
free, open source Core version as well as a closed source paid variant.

5.3.12   Pull request analysis

The selection of the pull requests for our analysis was done by directly using GitHub's interface. More
specifically, we selected the 10 most discussed accepted pull requests. Apart from the number of comments,
we also considered selecting pull requests that were submitted by contributors that are not Red Hat employees
in order to get a better overall view of the integration procedure.

A more detailed analysis can be found in the Appendix A.

5.3.12.1   Integrators

By analyzing[2] the most discussed pull requests we were able to identify the integrators of the Cockpit-
project. These are mainly:

  • Martin Pitt ([@martinpitt](https://github.com/martinpitt))
  • Stef Walter ([@stefwalter](https://github.com/stefwalter))
  • Dominik Perpeet ([@dperpeet](https://github.com/dperpeet))

5.3.12.2   Most discussed accepted pull requests

Every pull request undergoes a reviewing phase in which a lot of back and forth communication between
the assignee and the reviewers regularly occurs. Particularly, there are different reviewers for pull requests
affecting the functionality and different for those that update the interface of the system. To our understanding,
the reviewers focus mainly on the style and quality of the code and examine whether or not it corresponds to
the desired functionality. However, due to the CI pipeline, the implementation is automatically tested in
different stages. Furthermore, it goes without saying that every pull request should comply to the project's
design principles.

It also interesting to note that in contrast to a typical pull request, where the assignee usually provides the
entire implementation, in Cockpit there are parts of the desired functionality that are decided on the fly
during the discussion that takes place between the participants.

Another thing that we observed is that it might take a long time until a pull request is merged. Most often, the reasons behind this are either because of dependencies to other issues/pull requests or because of the recurring reviews and fixes between the participants.

5.3.12.3 Most discussed rejected pull requests

Unfortunately, we were not able to identify any rejected pull requests in the project's repository. We suspect, that the reason behind this is mainly because the majority of the contributors are all Red Hat employees. That means that the issues to be resolved are usually pre-discussed among the members of the team, thus always leading to a partial or complete merge of the pull request's implementation. This sounds reasonable since in comparison with more generic-nature projects, the highly specialized context of this project is not attracting many external developers.

5.3.13 Power-interest diagram

As project acquirers, RedHat, and therefore IBM have the power to be of great influence if they want to, but this will most likely not happen as they have relatively little interest in this project, since Cockpit is only a small, optional component of RHEL.

The core RedHat team is highly involved in development and they also make decisions on which direction the project goes. As their team leader, Stef Walter stands slightly above them in both power and interest. The contributors that do not belong to the RedHat team still have the possibility to influence the project by making pull requests, but the decision on whether or not to accept those PRs is ultimately that of the core team.

Linux distribution developers may have a slight interest in the project, since many distributions ship Cockpit in their default repositories. This also gives them a small bit of power, since they could threaten with removal from the default repositories if they really don't like a certain change. Browser developers generally do not use Cockpit or care about it, but browser compatibility is important for Cockpit's UI, so they should still be monitored.

Users hold quite a bit of interest, since they are the ones who actually rely on Cockpit on a daily basis. They also have a bit of influence since they can request features, and if enough users request a feature, the developers will tend to listen to that. Competitors are also interested in Cockpit since they have to compete with it. They generally don't have any influence, except maybe when they add a feature that is so important that the Cockpit team will be highly motivated to add that feature as well to keep up with them.

5.4 Context View

5.4.1 System scope and responsibilities

Cockpit is defined on their website as:

> The easy-to-use, integrated, glanceable, and open web-based interface for your servers
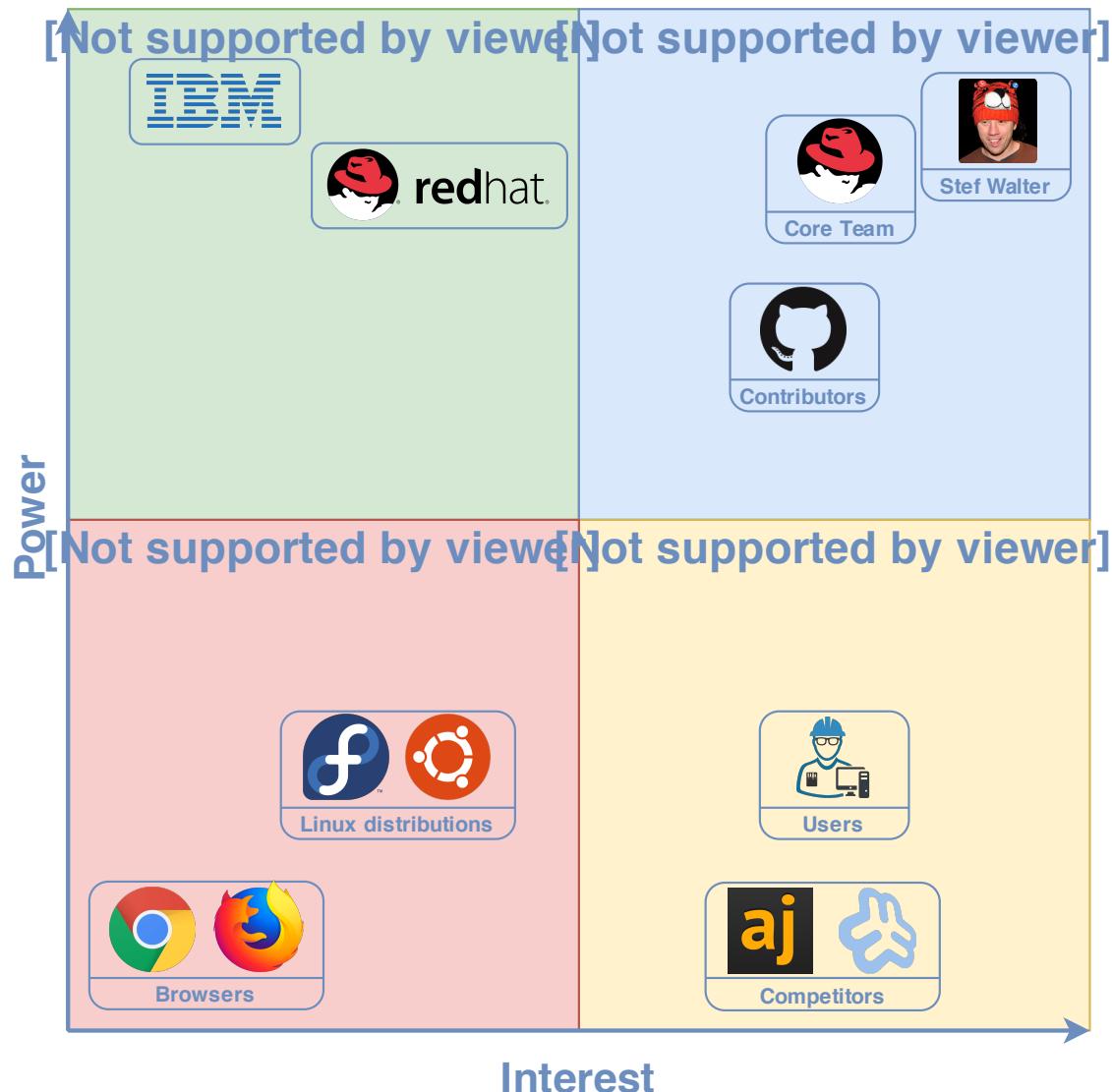
Figure 5.1: PI Diagram

It is a clickable interface for a variety of Linux distributions. Administration tasks that normally would have to be executed via the command line are now presented as a html page readable by your web browser. The GitHub repository of Cockpit gives a more detailed explanation:

> Cockpit is an interactive server admin interface. It is easy to use and very lightweight. Cockpit interacts directly with the operating system from a real Linux session in a browser.

System responsibilities:

- Enables managing of multiple servers in one Cockpit session
- Offers a web-based shell in a terminal window
- Supports efficient management of system user accounts
- Containers can be manged via Docker
- Provides a system overview for the health of the servers
- Diagnoses network issues
- Spot and react to misbehaving virtual machines
- Users can inspect SELinux logs and fix common violations in a click
- Supports Kubernetes or OpenShift v3 clusters
- Allows modification of networks settings
- Possible monitoring via the phone's browser

5.4.2 Diagram

The diagram below shows the context in which the Cockpit project exists.

5.4.3 External entities

From the diagram, we can observe a few external entities:

- Cockpit is mainly coded in JavaScript and C, with Python mainly being used for testing and automation scripts.
- Cockpit is published under the LGPL v2.1 license.
- Cockpit can be used to manage server hardware and VMs, but also supports Kubernetes, Docker and OpenShift containerized instances.
- The Cockpit UI is rendered inside browsers, with Chrome, FireFox, Edge, Safari and Opera being officially supported.
- The Cockpit backend can run on a multitude of OSes, with RHEL, Fedora, Ubuntu, CentOS, Project Atomic and Debian being listed as fully supported.
- The Cockpit test suite relies on Semaphore for CI, as well as Selenium and Avocado for integration tests.
- The code of the Cockpit project is hosted on GitHub, which is also used for version control and issue tracking.
- Freenode is used for communication via IRC
- Cockpit's main competitors are Ajenti, Webmin and Nagios.
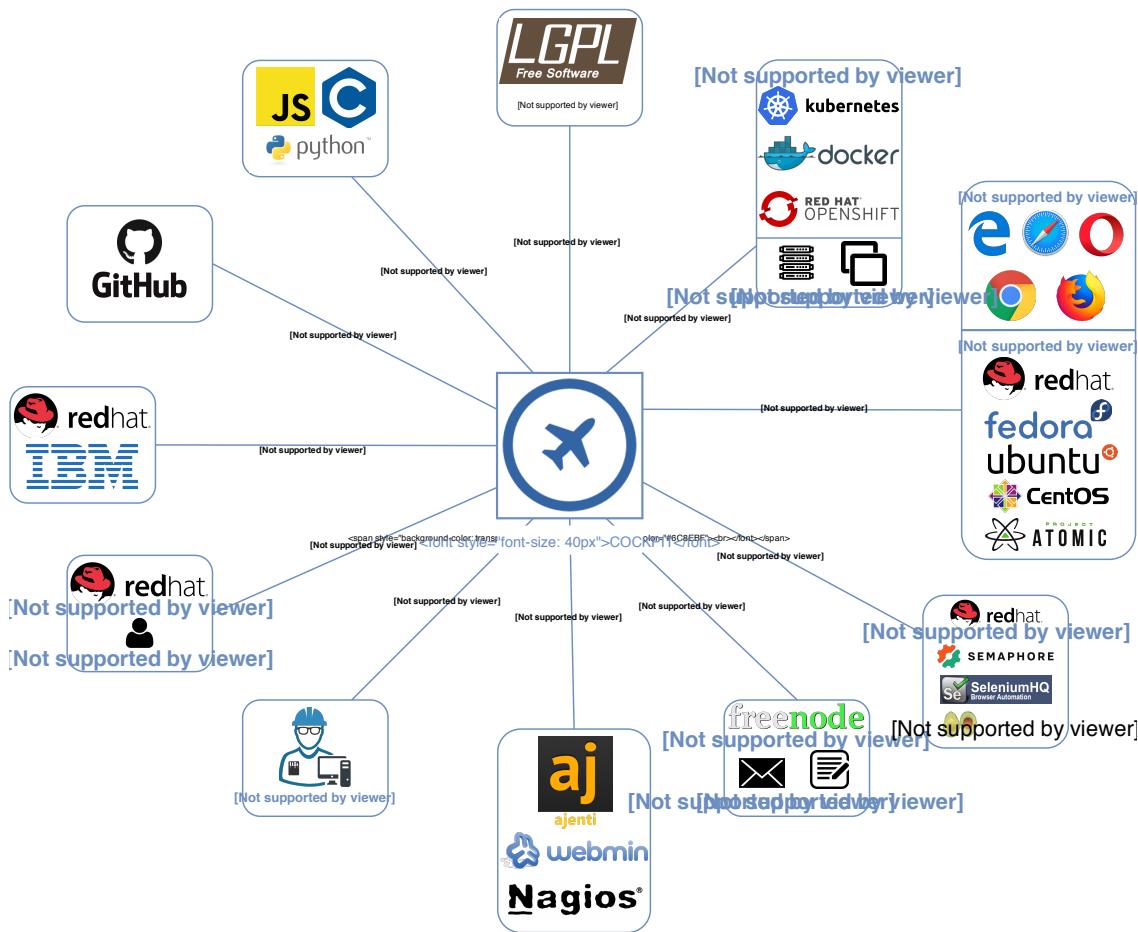
Figure 5.2: Context View

5.5 Development view

The architecture that is specific to the development process is described in this section.

5.5.1 Module Organization
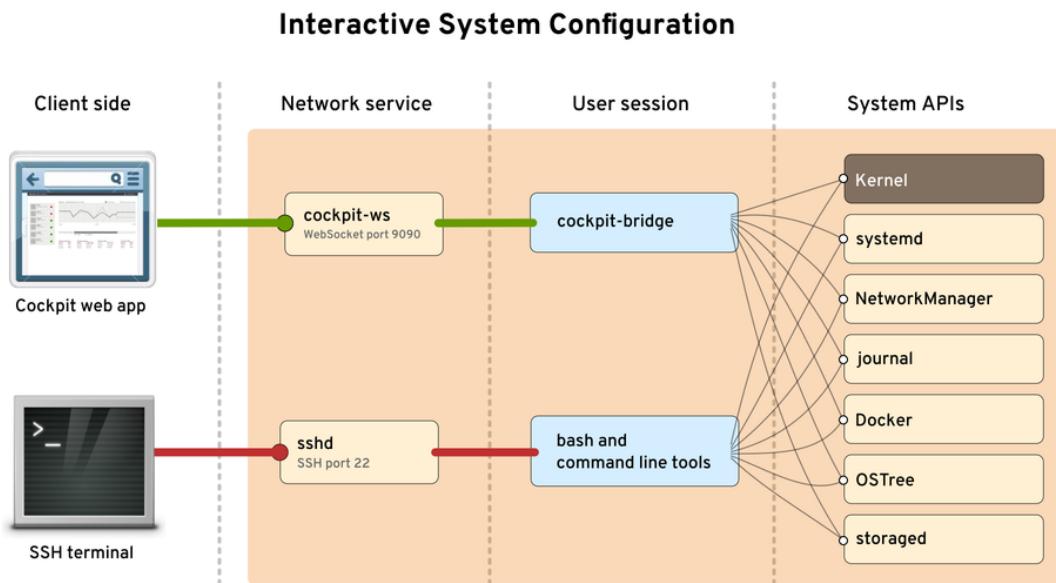
The main components described by the manual pages on the Cockpit website are:

```
cockpit.conf -- Cockpit configuration file
cockpit-ws -- Cockpit web service
cockpit-desktop -- Cockpit Desktop integration
remotectl -- Remote Access Configuration
cockpit-bridge -- Cockpit Host Bridge
```

These components don't naturally map to the source code in the GitHub repository, because they are targeted towards users, not developers. A total of 12 directories are listed on the GitHub repository. The source code is written in two main languages: C and `javascript`. These two languages respond to the directories `src` and `pkg`, respectively. The 'Contribute' page explains:

> Most of Cockpit is written in JavaScript. Almost all of this code is found in the packages in the pkg/ subdirectory of the Cockpit git checkout.

There is also a directory for tests called `test` and a directory for bot functionality called `bots`.

As can be seen in the picture above, retrieved from the Cockpit docs[3], the user interacts via the Cockpit websocket. The websocket is coupled with the API via the Cockpit bridge. The API is what is interesting for a developer and provides most of the functionality of the application. A summary of the API follows, mostly deduced from the Cockpit docs.

- **systemd** provides configuration and monitoring capabilities with the system (OS)
- **NetworkManager** provides interaction with the systems network configuration
- **Journal** provides indexed log data
- **Docker** provides Docker container management functionality
- **OStree** provides versioning updates of Linux-based operating systems.
- **storaged** provides disk storage and mounting and other such functionalities

5.5.2   Common Design Model

Cockpit consists of packages that contain components which are HTML documents. The Cockpit documentation is quite extensive on the subject of packages and URLs. Packages are standardized to integrate easily into the already existing code base. Special URLs are used to refer to internal components.

5.5.2.1   Design Standards

For software projects with different contributors, it is necessary to establish some design standards that all developers must adhere to, in order to keep the code clean and comprehensible for all developers. To achieve this, the Cockpit core team has established the *Cockpit Coding Guidelines*, which can be found on the GitHub wiki. In these guidelines, standards are listed for general coding style, such as spacing and indentation, with language-specific guidelines for C, JavaScript, CSS/HTML and Python.

Furthermore, there is also a section on design guidelines for the GUI style based on PatternFly, which can be found on the PatternFly website. These standards are used to ensure the GUI style is consistent across the entire program, and also saves developers the hassle of worrying about GUI-related design.

5.5.2.2   Common Processing

Throughout the codebase, data is usually transported from one location to the other using the `DBus` interface, which is implemented in JavaScript. `DBus` is, as the name suggests, a software bus to allow for inter-process communication. It was developed by RedHat, which explains its presence in the RedHat-dominated code of Cockpit. The Cockpit developers mostly use it to make the Cockpit web app GUI communicate with the backend. It can be used by calling the `cockpit.dbus()` function. Developers can specify which `DBus` service they wish to connect to via the function argument, and the function returns a service variable. This service can be interacted with using its `.proxy()` function, which can be used to send or receive data.

5.5.2.3   Package Layout

A package must consist of one or more files and include a `manifest.json` which provides metadata and has a number of required fields. The name of the package is equivalent to the name of the directory it is located in. Naming conventions are employed for all files. Referring to other packages (linking) is done relatively. The `cockpit-bridge` is used to interact with the front-end of the application.

5.5.2.4 URL Specification

URL addresses are always relative, even if they are used to refer to resource in other packages. All URLs are in the `/cockpit` namespace and thus start with this prefix. Following the namespace identifier is either a host or checksum, actually telling where to find the package. Next is the package name, the component name and a hash, which allows to navigate within a single component. The URLs are not visible to the end user by default. URLs are wrapped to provide bookmarking capabilities and other standard browser features. If the path of a URL does not exist, users are redirected to the default page or dashboard.

5.5.3 Codeline Models

The root directory of the Cockpit project contains 12 directories. The most important directories will be discussed here. The release process, and continuous integration are also analyzed.

5.5.3.1 Resource Structure

The packages that are the main part of Cockpit can be found in the `pkg` directory. The `src` contains mostly inward facing functionality, which needs to be fast and is therefore written in `C`. The `bots` directory is related to automated testing and consists of OS images, virtual machines, overrides, and tasks. Tests are located in the directory `test` and make use of the resources in th `bots` directory. Before running the integration tests, the `image-prepare` script in the bots directory needs to be executed. Documentation such as markdown, images and xml is located in the `doc` directory. The `po` directory contains translations for multiple languages in `.po` format.

5.5.3.2 Release Process

Cockpit has had 189 releases since its creation. The major releases are sequential, but sometimes patches for previous releases are submitted. For every major release, a blog post is made on the Cockpit website discussing the newly added features. The posts are often illustrated with pictures to make improvements more tangible.

Almost every two weeks a new release is made. Releasing within short intervals reduces the amount of introduced changes. Moreover, it shrinks the management overhead and support duration. The short release cycle maps to the AGILE SCRUM process, clearly indicating milestones of the project. Contributors can see their code being used in production quite fast. The quick feedback of bugs or other faults helps to keep the code base healthy.

5.5.3.3 Continuous Integration

Integration testing is done via a few scripts residing in the `test` and `bots` directories.

The first step is to verify that the build is in the expected location and built correctly. This can be done using the `./bots/image-prepare` Bash script. After that, the automated test suite can be activated using the `./test/verify/run-tests` Bash script.

These tests are used to check integration of the core program with multiple OSes. The test settings can be changed using environment variables. * `TEST_OS` sets the operating system(s) to be tested, with multiple architectures of CentOS, Debian, Fedora, RHEL and Ubuntu being currently available. * `TEST_DATA` is used to set the OS VM image storage location * `TEST_JOBS` is an integer setting to change the amount of tests to be ran concurrently. * `TEST_CDP_PORT` can be used to automatically attach to a browser that supports the Chrome Debug Protocol. This setting sets the port used for CDP.

The UI integration is tested using Avocado (`./test/avocado/run-tests`) and Selenium (`/test/avocado/selenium-base.py`). Selenium supports Chrome, Firefox and Edge browsers. The testing browser is selected by setting the `BROWSER` environment variable.

The aforementioned test scripts are written in Python3, and requires both `selenium` and `avocado-framework` packages to be installed.

The `README.MD` of the repository shows the Semaphore build status. When clicked, it redirects to a page where the project's build status can be monitored in real-time.

5.5.3.4 Folder structure
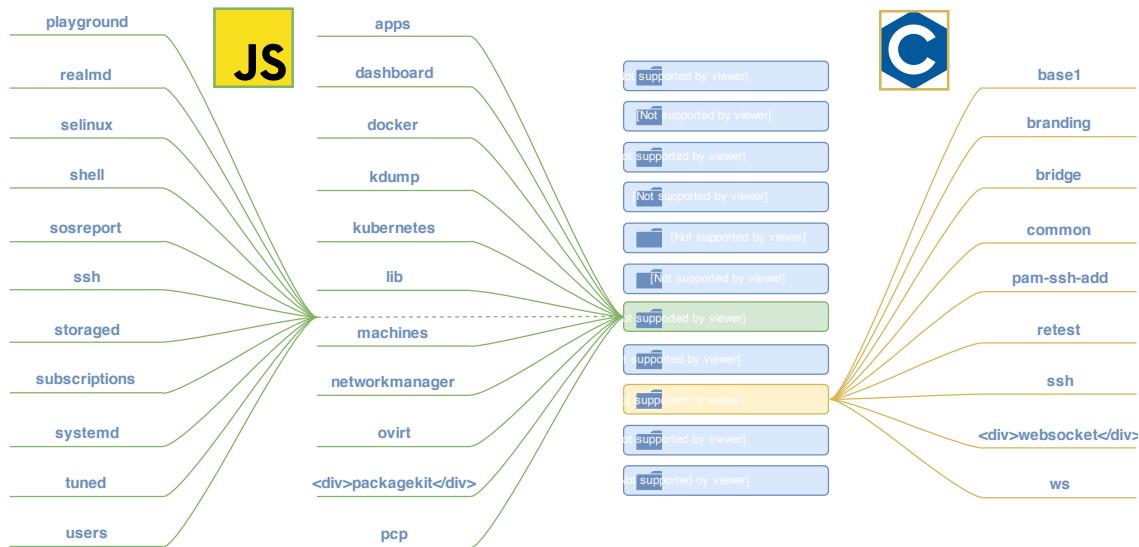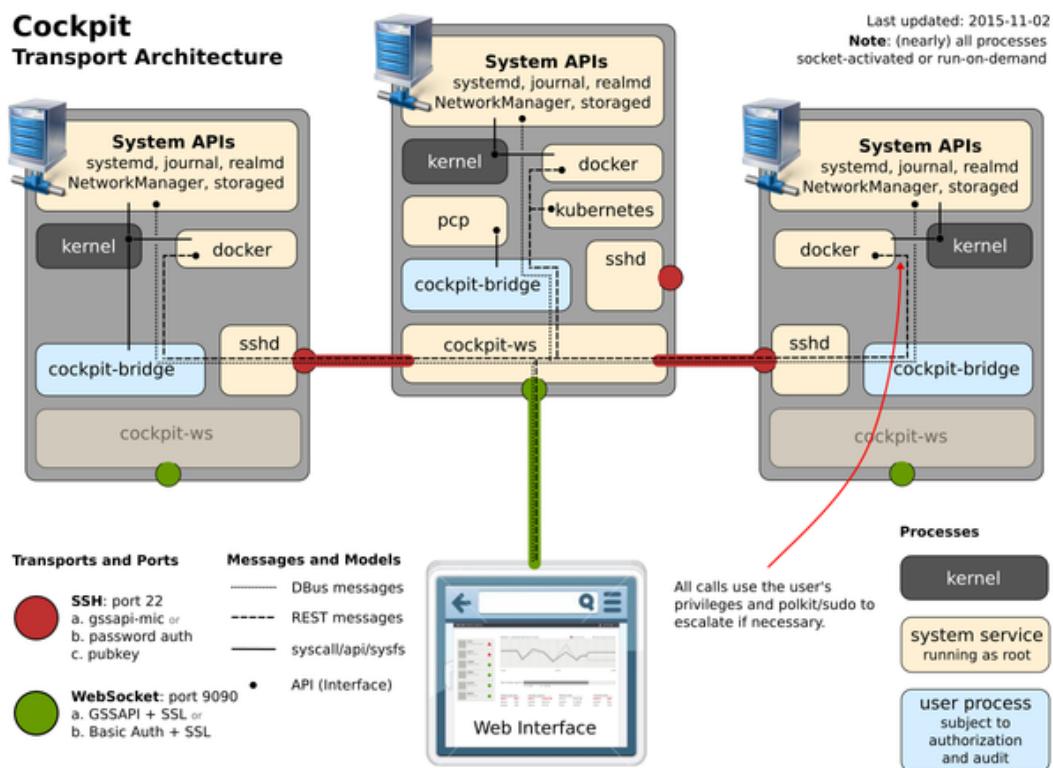
Figure 5.3: Folder structure

5.6   Deployment View

The Deployment view describes the view of the system after it is built and tested and is ready to be deployed to production. In this section, we examine some particular aspects of the Deployment view in Cockpit.

Cockpit is implemented for **standalone usage**, meaning that it does not follow a client-server architecture. On the contary, cockpit is deployed on top of the particular operating system that the user is currently using and runs directly in the client's browser. Furthermore, each action at the interface is translated to a linux command, hence there is no need for a database since no intermediate information is stored. On the other hand, even as a standalone application cockpit can connect and interact with remote nodes/containers or even with entire **OpenShift** and **Kubernetes** clusters. This is done the same way as with an SSH session, and the users can troubleshoot, configure and interact with the remote system using the web UI of cockpit deployed on their local machine. For that to happen, each of the remote Linux instances needs to have Cockpit installed. The following image shows an overview of this procedure:

As can be seen in the image above taken from the project's documentation, Cockpit connects to the various system APIs using the **cockpit-bridge** module. This bridge allows the sending of messages and commands from the Web front-end to the server. The **cockpit-ws** (Websocket) program enables the communication between the web browser and various APIs and components such as cockpit-bridge. In this diagram, through cockpit-ws, a connection is established over SSH with two remote server instances.

Web Browsers: Cockpit can be accessed by most popular web browsers. The minimum required versions are the following:

- Mozilla Firefox 52
- Microsoft EDGE 16
- Google Chrome 57
- Apple Safari 10.3
- Opera 44

Operating systems: Cockpit is included in the major Linux distributions. In detail, cockpit supports the following operating systems:

- **Fedora** : Cockpit is installed on the Fedora server. No version limitations are specified.
- **Red Hat Enterprise Linux**: Red Hat Linux Extras Repositories (versions 7.1+) include Cockpit.
- **Project Atomic**: Cockpit can connect to to an Atomic host via the user interface. No version limitations are specified.
- **CentOS**: Cockpit is supported in CentOS 7.X.
- **Debian**: Cockpit is included in Debian unstable and in Debian 8, 9 backports
- **Ubuntu**: Cockpit is supported in Ubuntu 17.04 and later. It is also available as an official backport for 16.04 LTS and later.
- **Clear Linux** : Cockpit is included in Clear Linux. No version limitations are specified.
- **Arch Linux**: Cockpit can be found as a package in the Arch Linux repository. No version limitations are specified.

Hardware Requirements: No hardware minimum requirements are specified to deploy Cockpit. This is quite unexpected since from our experience Cockpit was quite slow when we deployed it on our local VMs. On the other hand, Cockpit claims to be **zero footprint** meaning that when the system is not used, no resources are being allocated.

Nevertheless, even if the exact requirements are not mentioned, we can make an estimation of the required resources by looking at the specifications of the VM that the team uses to run the integration tests. Our investigation showed that this is a *x86_64* system with *4 cores* and *2GB* of memory. To that end we can assume that any system with similar or better hardware components will be capable of running cockpit. We also tested it on a system with 2 cores, it showed no slow down and performed similarly.
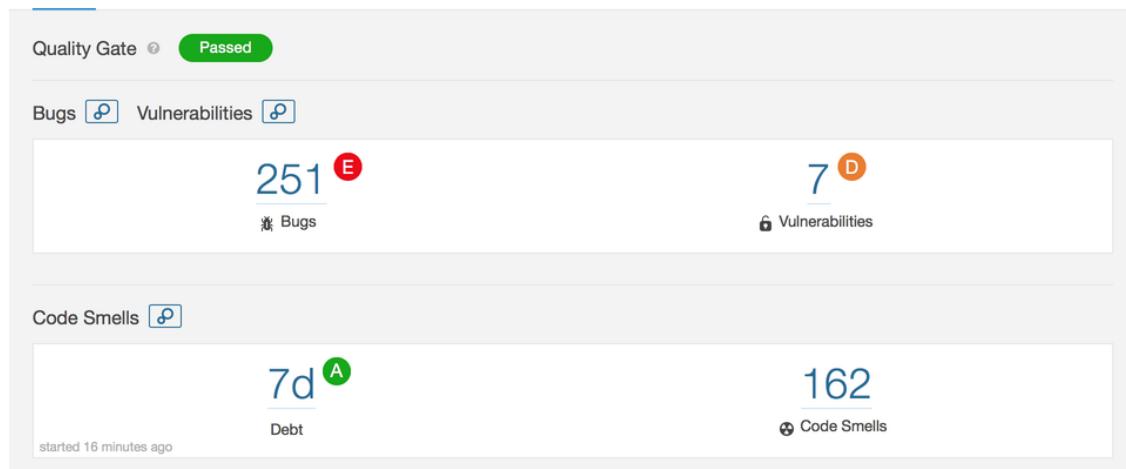
5.7 Technical Debt

In this section we describe our findings in regards to Technical debt. Specifically, we attempted to identify Technical debt both manually and through the use of static analysis tools.

5.7.1 Static Analysis
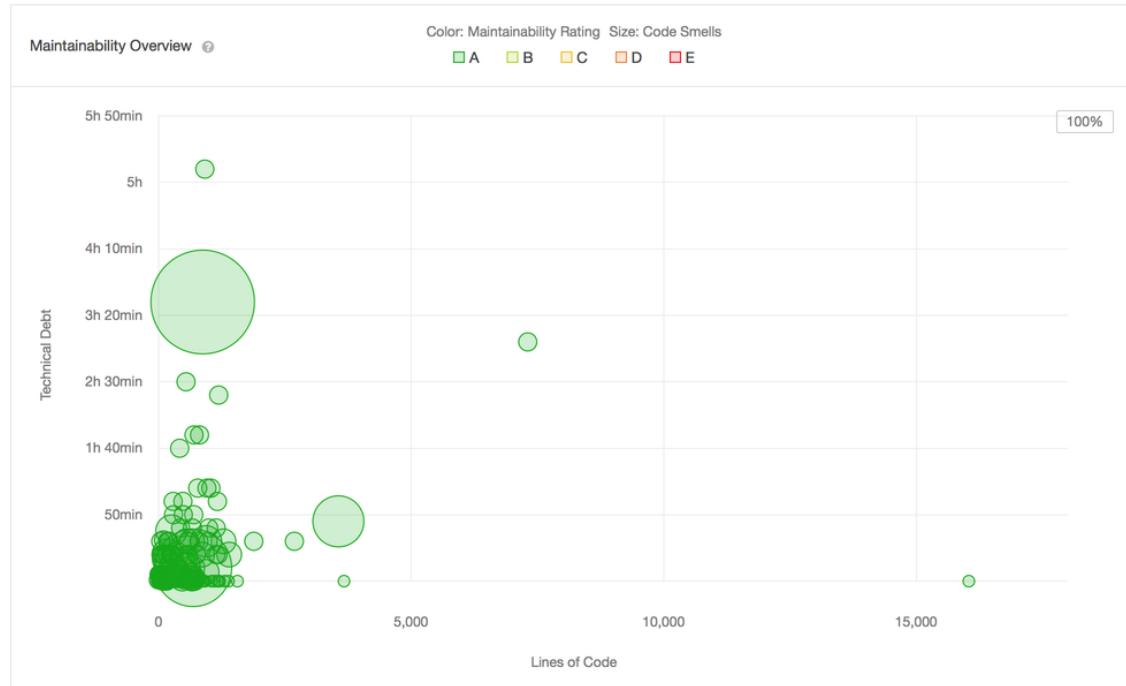
To perform static analysis on Cockpit, we used SonarQube with support from FlawFinder and CPPCheck for a more in-depth analysis of Cockpit's central source code.

SonarQube: SonarQube analyzed a total of 187.000 lines of code, most of which were composed of JavaScript (87.000) and C (75.000). Of the 251 bugs detected, almost all were generated by deprecated HTML elements and improper tag usage. All exceptions to this rule were attributed to JavaScript testing files.

Vulnerabilities in Cockpit mainly originate in the `pkg/shell` and `pkg/kubernetes`. The majority of them have to do with JavaScript code not checking that a sent message is successfully received. Analysis of code smells revealed that most came from core source files in `src/bridge` (2d, 4h). Test files were the largest offender, and were mostly generated because of useless assignments.

According to SonarQube, the estimated time required to address the technical debt from code smells is 6d 4h. The maintainability figure below visualizes the impact by file:

Finally, SonarQube assigns Cockpit a cyclomatic complexity score of 16.916. Notable offenders from the `src` directory were `base1/cockpit.js`, with a score of 1222, which is close to half of the entire score of src (2623). The worse though came from the `pkg` directory, with a score of 14.249.

FlawFinder: The FlawFinder tool searches for security vulnerabilities (e.g buffer overflow risks, race conditions etc.) and produces a report with all findings categorized by risk level.

We deployed FlawFinder on the `src` directory of Cockpit, with the produced results summarized as follows:

Vulnerability Category	Count
Race	9
Buffer	287
Shell	18
Format	4
Crypto	2
Random	6
Integer	2
Access	2
Misc	57

Of the 102.755 lines analyzed, there were five vulnerabilities with level 5 (the maximum level). When examining the details of the report concerning the maximum level vulnerabilities, we found that 4/5 are from test files and are related to race conditions.

CPPCheck: CPPCheck is a tool which primarily searches for bugs and dangerous coding constructs in
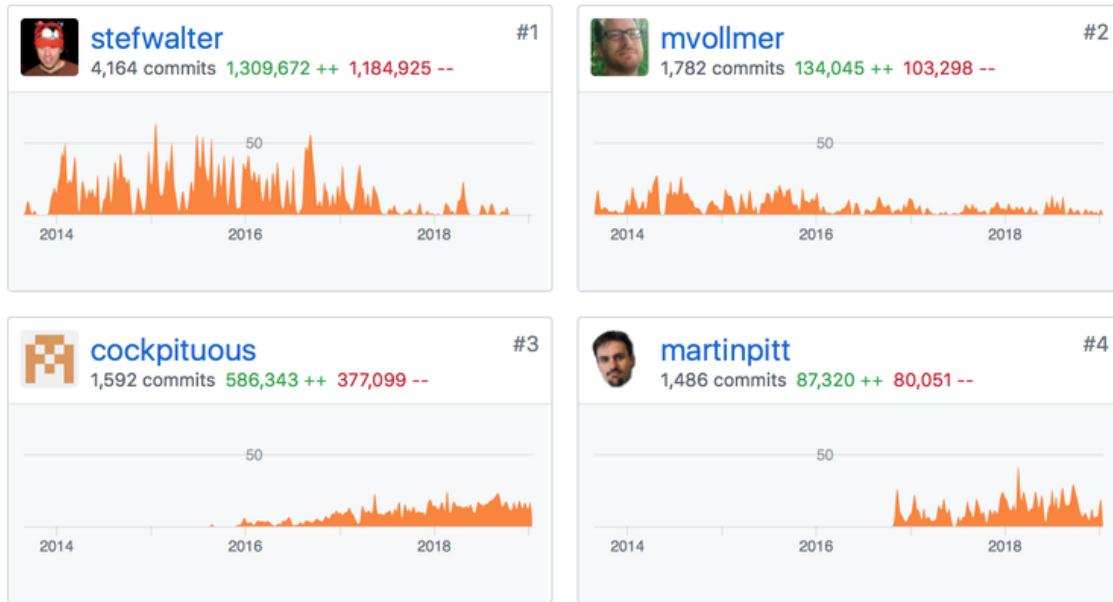
C/C++ code. Like with FlawFinder, we deployed it on the `src` directory of the Cockpit project and collected the following statistics:

Category	Count
Unused Function	309
Variable Scope	268
Redundant Assignment	138
Invalid Scanf	1
Variadic Function NULL UB	85
Null Ptr Redundant Check	12
Literal Char Ptr Compare	38

Some of the less obvious bad practices would be passing `NULL` as the last argument to a variadic function in C, and using `sscanf` without field width limits. Others are more or less harmless, such as a variable having a larger scope than it needs. It is important to mention that a decent portion of the flaws detected by both CPPCheck and FlawFinder originate in the test files that litter the source code directories. This means that some of the vulnerabilities mentioned by FlawFinder would not be a threat in production.
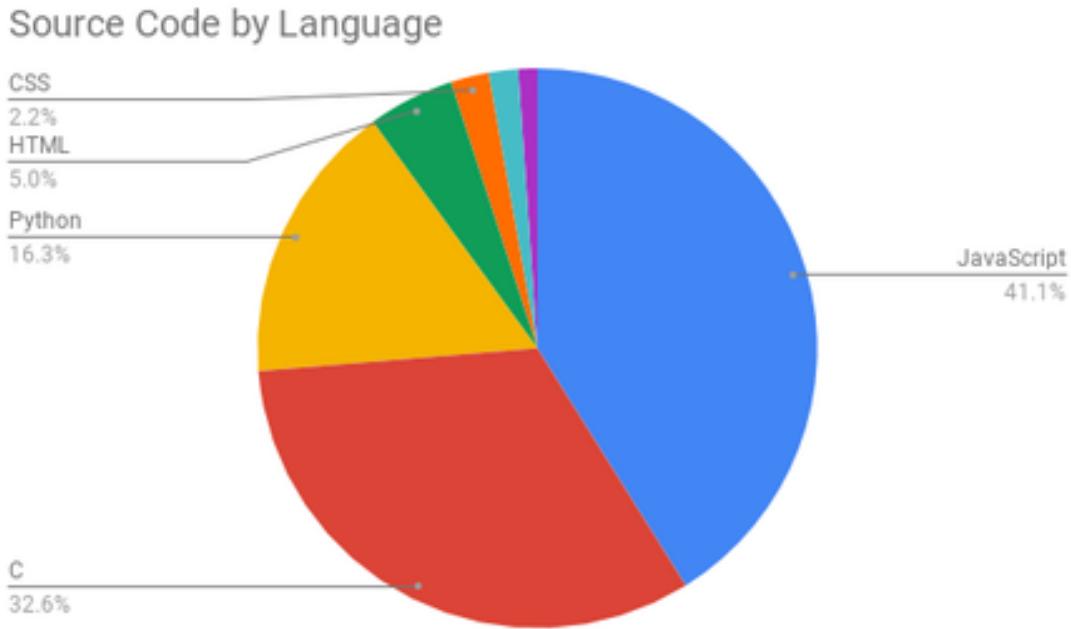
5.7.2 Bus Factor

The main contributions to the Cockpit project come from a small party of 3-4 individuals. In fact, GitHub's metrics indicate that the leading contributor ([@stefwalter](https://github.com/stefwalter)) is the largest contributor by a wide margin. Although we were expecting that most contributions would be concentrated in a smaller subset of developers, the fact that Cockpit deploys a substantial amount of sophisticated low-level infrastructural code suggests that the bus factor for the project may not be entirely insignificant. Even being open source and well commented, the use of intricate channeling infrastructure provides a high barrier to entry, requiring a developer to have an in-depth understanding of not only Cockpit's plumbing, but the operating system functions it works with.

5.7.3   Design Debt

The infrastructure driving Cockpit is primarily written in C. This is presumably done because of the language's excellent interoperability with Linux and its familiarity to the core developers. The project also makes use of an abundance of specialized low level infrastructure. JSON parsers, websockets, authentication, pipes, and daemons hooked to `systemd` are all written from the ground up for use specifically with Cockpit. This infrastructure allows different components of Cockpit to communicate efficiently and helps abstract the details of passing information around from the rest of the code. Although it provides the developers with a high level of control, it also imposes a considerable design debt in terms of maintenance. Extending the source now requires intimate knowledge of its plumbing. This burden now becomes the responsibility of the developers.

Source Code by Language

CSS
2.2%
HTML
5.0%
Python
16.3%

JavaScript
41.1%

C
32.6%

5.7.4 Testing procedure and observations

Cockpit uses Semaphore as their CI platform, accompanied by automatic testing frameworks/tools such as Avocado and Selenium. These along with the cockpituous module which is described as "A fleet of robots that run the test suites for each pull request" provide a quite thorough testing procedure. It is worth to note that this procedure includes the building of **26** distinct VM images on which the tests are being run.

In order to identify the code coverage of the existing tests we tried executing the tests manually and looking at their Semaphore CI page. Unfortunately, although the former outputs the number of tests that are being run, neither showed an exact percentage of the code covered. Albeit the team probably has some internal tools that show this metric, we consider this to be extremely inconvenient for new contributors.

There are 2 types of tests in Cockpit: **unit tests** and **integrations tests**.

The first category consists of 2914 tests that focus on testing the functionality of the code. Furthermore, since Cockpit is a project that relies on Linux OSes and supports a big variety of browsers, it is important to guarantee that code updates do not break the existing functionality. That said, integrations tests are essential for this project since they ensure cross-browser compatibility and that new additions are platform independent.

5.7.4.1 Improvements

We propose two major additions that we believe would contribute to the project's stable future and encourage more developers to contribute. Firstly, we observed that Cockpit is quite slow. To that end, we propose the

addition of **performance tests** which would help identify the bottlenecks of the project and prevent future additions that would further decrease the performance. Moreover, the percentage of code coverage should be available to the entire community and not just the core RedHat team. Finally, it's quite hard for a new contributor to understand their testing procedure at a glance. That is because the instructions cannot be found in a single concrete document but are separated in different files/web-pages.

5.7.5   Technical debt evolution & awareness

On top of the aforementioned analysis, technical debt awareness of the developers is examined in this section.
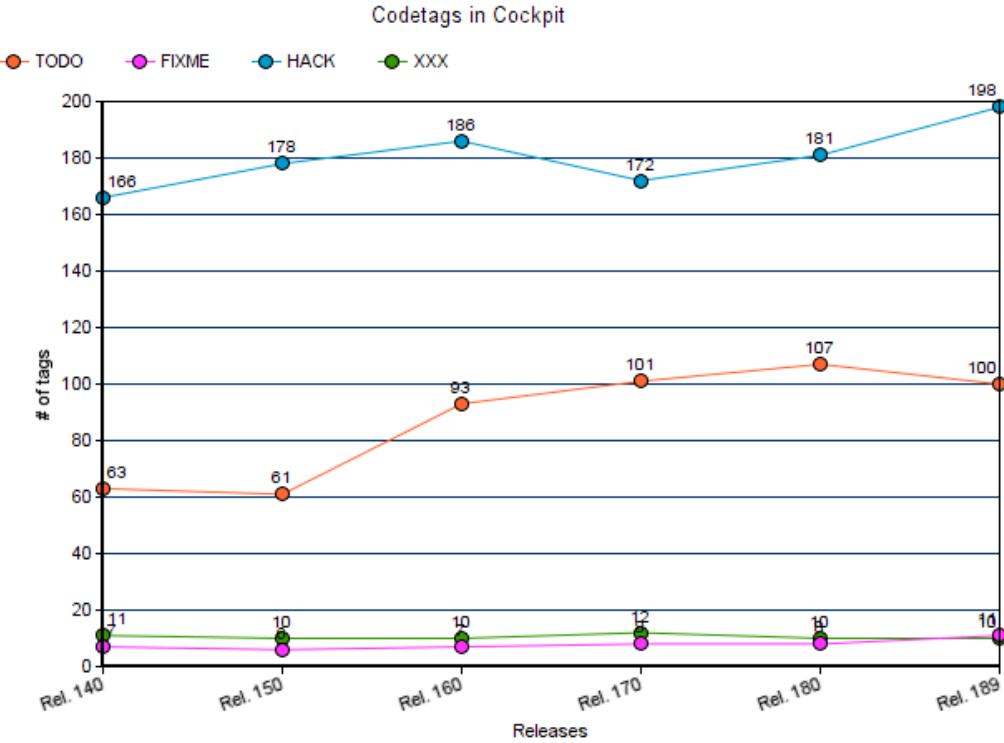
We did this by looking into the following two aspects:

- The discussion between the developers for particular issues
- Specific codetags (TODOs, FIXMEs, HACKs, XXXs) which are generally used as reminders of parts of code that need closer inspection or review.

Regarding the first aspect, we observed that for complex issues, the involved developers discuss how the implementation should be so that it will affect the maintainability of the project. Moreover when it comes to code reviews, the reviewers tend to suggest changes that will optimize the code, regardless if it already works. In regards to the second analysis, our codetags lookup on the latest release (189), showed some interesting insights. In particular, multiple codetags were found as shown in the table below. A more detailed analysis can be found in the Appendix B.

Codetags	Count	Files
TODO	100	66
FIXME	11	10
HACK	198	112
XXX	10	5

These results indicate that the developers are aware of the technical debt in their project since they annotate their code with tags that imply the need for further investigation. Moreover, in order to identify whether the developers address the issues described by the tags or they just ignore them, we compared the latest release with 2-year old past ones.

Codetags in Cockpit

As can be seen, there is a undoubtedly a increase in the number of codetags that need to be addressed. This leads us to the conclusion that although the team seems to be aware of the technical debt of their project, minor efforts are being done in order to decrease it.

5.8 Conclusion

Throughout these sections we examined various aspects of Cockpit's architecture. In particular, we made an extensive stakeholder analysis in order to identify the main stakeholders of the project, we examined various viewpoints of the system and analyzed its evolution and technical/testing debt. Our analysis lead to some interesting insights in regards to the underlying architecture of Cockpit and open-source projects in general.

Early on during our stakeholders analysis we identified that the majority of the stakeholders are Red Hat employees. This was quite surprising considering the size of the project but was later justified by observing that there is only a small number of active contributors. Furthermore, by analyzing the different viewpoints we learnt how Cockpit is deployed and developed and what are its dependencies. The results of that analysis indicate that Cockpit's codebase and testing procedures are quite complex and that partially explains the reasons behind the small number of non-RedHat contributors. Lastly, our technical and testing debt analysis showed that although minor, there is technical and testing debt in the project. Judging by the significant number of code tags that were found in the source code, we argue that the developers-team seems to be

aware of that. However, by comparing the latest release with previous ones we were able to find that not much effort is spent on addressing these issues.

In summary, our findings allow us to argue that Cockpit is a promising and undoubtedly useful for Linux system administrators. However, we believe that there is still room for improvement in order for the project to evolve at a faster pace.

5.9   References

[1] https://www.gnu.org/licenses/old-licenses/lgpl-2.1.html

[2] Gousios, G., Zaidman, A., Storey, M. A., & Van Deursen, A. (2015, May). Work practices and challenges in pull-based development: the integrator's perspective. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1* (pp. 358-368). IEEE Press.

[3] https://github.com/cockpit-project/cockpit/tree/master/doc

5.10   Appendix A

5.10.1   Most discussed accepted pull requests

Name	Context	Merged by	Issued,Merged	Key points
machines: Add create vm dialog #7820	Refers to the creation of virtual machines through the interface	Martin Pitt	**Issued:** 05/10/2017; **Merged:** 02/02/2018	- Was reviewed by many individuals including designers and senior developers; Request for fixes: Back and forth communication between the assignee and the reviewers; Merge delayed due to multiple rounds of reviews and fixes; 4 months passed until the final merge

Name	Context	Merged by	Issued,Merged	Key points
test: Stop reloading sshd when Debian networking changes #6158	Disable broken behaviour caused by sshd reload	Martin Pitt	**Issued:** 20/03/2017; **Merged:** 22/03/2017	- The pull request was prioritized since it was blocking other components; Due to the prioritization, only 2 days were needed for it to be merged
cockpit:machines: Display VM graphics console #5932	Graphics console retrieval for a VM	Martin Pitt	**Issued:** 23/02/2017 ; **Merged:** 08/06/2017	- Discussion about whether to use VNC or SPICE to address the issue; Consulted individuals that have used SPICE in their own projects; Merge was delayed due to the discussion regarding the available choices and due to the extra implementation that was needed
vms: VM management - initial commit #4434	VM management and monitoring	Dominik Perpeet	**Issued:** 18/05/2016 ; **Merged:** 29/09/2016	- Was blocked/delayed due to dependencies to other issues; Many rounds of reviews and fixes; Many rebases on top of the existing implementation

Name	Context	Merged by	Issued,Merged	Key points
Navigation updates (CSS for desktop and the start of mobile) #7482	Visual navigation updates for the desktop and mobile interfaces	Martin Pitt	**Issued:** 10/08/2017; **Merged:** 13/11/2017	- Included an extensive design related discussion; Required multiple tests to ensure consistency among the different browsers; Again submitted partial implementation and added more after discussions
Adjust RHEL 7 images for the 7.5 release #8984	Prepared tests for the new RHEL release	Marius Vollmer	**Issued** 12/04/2018; **Merged** 13/04/2018	- The contributor of this PR was team member of the project; An adjustment in the RHEL 7 images was needed to support the new release; Two different commits were referenced as guidelines and after it passed the tests it got merged in the master

Name	Context	Merged by	Issued,Merged	Key points
docker: implement new design #4952	New design for docker page	Lars Karlitski	**Issued** 01/09/2016; **Merged** 19/10/2016	- An implementation for a new design of docker was needed; The PR had some unrelated test failures in different modules, including `rhel failure` and `testRestart` in fedora 25; In the end some fixes where required and the implementation was force pushed to another branch
Add CentOS CI image store #9321	Update CentOS image store pertaining to ubuntu and REHL	Martin Pitt	**Issued** 06/06/2018; **Merged** 11/06/2018	- The initial update failed the tests so the contributor had to fix bugs that were causing the failures; The PR finally merged, even though they were some unrelated test failures; The Cockpituous bot is employed to automatically update the images

Name	Context	Merged by	Issued,Merged	Key points
systemd create timer option #4645	Functionality to execute shell commands at a specified time	Harish Anand	**Issued** 28/06/2016; **Merged** 14/08/2016	- A PR made by a CS graduate - The PR is reviewed by a member of the core team for code style and design; The changes are accepted and it is stated that it will be part of Cockpit 0.118.
cockpit:machines: Display VM graphics console #5932	Desktop-like server interaction within browser iframe	Marek Libra	**Issued** 23/02/2017; **Merged** 08/06/2017	- Cockpit already supports non-graphics console; The original post of the PR explains the new feature and the number of different implementations possible. It also shows a video of the new feature; Developers of the graphics consoles join the discussion; Multiple implementations are required, both in-browser and not in-browser to best suit the needs of all users

5.11   Appendix B

A detailed analysis regarding the codetags per package can be seen below. Note that trivial packages such as /examples and /doc have not been included.

	TODO	FIXME	HACK	XXX
/bots	0	1	37	0

	TODO	FIXME	HACK	XXX
/containers	1	0	7	0
/pkg	75	2	50	6
/src	15	0	34	0
/tools	1	1	1	0
/test	8	5	67	4

As can be seen in the table, a significant number of the codetags can be found in the /pkg and /src packages. This, is reasonable considering that these 2 packages contain the interface and core logic respectively. What is worrying though is that there is a big number of HACK tags in the /test package which indicates that some tests are not complete. On the other hand, each of these tags corresponds to an existing issue which is tracked in Redhat's bugzilla issue tracker. An example of such an issue is the following:

```
# On Atomic no locales other than en_US are supported on the host itself
# HACK: https://bugzilla.redhat.com/show_bug.cgi?id=1186757
```

Chapter 6

Django

By Alexandru Balan, Andra Ionescu, Phil Misteli, David Vojtek

6.1   Table of Contents

6.2   Introduction

According to the description on Django's website:

> Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design.

In other words, Django is an open source web framework that enables the user to write complete applications from scratch by offering support for building a back-end infrastructure, as well as creating the front-end interfaces and interactions through its templating engine. Its support for various databases and the ease of integrating it with various front-end frameworks make it extremely versatile for various use cases, such as content management systems, computing platforms or social networks.

This chapter aims to describe the high-level architecture starting with an analysis of the stakeholders, followed by the context and development view, technical debt and finally, the evolution of the project.

6.3   Stakeholders

According to Rozanski and Woods [1], a stakehold is a person, a group or an entity that has interest and concerns about the realization of the architecture. According to their classification based on the roles and concerns, 10 types of stakeholders are identified: acquirers, assessors, communicators, developers, maintainers, suppliers, support staff, system administrators, testers and users. In Django, the non-profit organization **Django Software Foundation** (DSF), the **Core Members** and the **Technical Board** have a major impact in the administration of the project. Thus, special attention is required in order to fully understand their responsibilities and how they integrate with the RW categorization.

6.3.1   Django Software Foundation

Based on the Django organization, the Django Software Foundation handles the financial and legal aspects of the project. The goal of the foundation is to promote, support and advance the project. Therefore, they support the development by organising meetups and community events, by promoting the use of Django in the online development community, protecting the intellectual property and advancing the state of the art in Web development. The foundation is composed of **corporate members**, **individual members** and the **board of directors**.

The corporate members offer the financial support in the form of one year subscription based packages. The packages are formed based on the number of people from the organization and the payment amount, ranging from organizations with up to 10 people and a dues of 2000$ (Bronze package) to organizations with more than 500 people and dues of 100,000$ (Platinum package).

The individual members are people appointed by DSF that deserve the recognition for their services in the community.

The board of directors is formed of Frank Wiles - President, Anna Makarudze - Vice President, James Bennett - Secretary, Jessica Deaton - Treasurer, Katie McLaughlin and Ola Tarkowska. Most of the members of the board of directors are public speakers, who help with Django events and conferences organization or are very experienced developers.

6.3.2   Core Members

The core team members are a group of volunteers that have shown dedication over time and became trustworthy to manage the Django Project. Some of the attributions of the core members are:

- triaging tickets;
- writing, reviewing and merging patches;
- managing the continuous integration infrastructure and the servers;
- participating in design decisions;
- handling the security issues;
- packing the release.

They also have authority over the Django Project infrastructure, the webite, the Github repository, bug tracker, mailing list and IRC channel. Based on the area of expertise, the core members are divided as follows:

- Ops Team - who maintains the infrastructure
- Releasers - who manages and handles the releases
- Security team - who is specialised in security matters and handles all the security issues reported
- Technical advisory team - veterans who are less involved in the day to day matters
- Technical board
- Technical team - Veterans contributors who are active in the day to day development

6.3.3 Technical Board

The Technical Board is a special category of core members. Their role is to steer the technical choices and to mantain the quality and stabilty of Django. The members are chosen on every release and the current board of the 2.2 release is: Adam Johnson, Andrew Godwin, Aymeric Augustin, Carl Meyer, James Bennett. The main attributes of the technical board is to grant or remove commit access and make decisions when no consensus is found.

Based on the three major categories idetified and their roles in the organization, the RW stakeholder categorization of Django is as follows:

Type & Description	Stakeholder	Observations
Aquirers *Oversee the procurement of the system or product*	Django Software Foundation	The DSF oversees the whole financial and legal aspects, but the funding, in particular, comes from the Corporate Members: Platinum (JetBrains, Instagram), Silver (Sentry, Cadre, Education Ecosystem and more), Bronze (Boomerang, Divio, Django Stars, TeamUp and many more)
Assessors *Oversee the system's conformance to standards and legal regulation*	Technical Board / Django Software Foundation	Technical Board maintains the quality and stability, while the DSF handles the legal regulations
Communicators *Explain the system to other stakeholders via its documentation and training materials*	Django Software Foundation / Core Team	DSF represents the communicators through the board of directors who are involved in organising conferences and events, while the core team helps explaining the system from a technical point of view
Developers *Construct and deploy the system from specifications (or lead the teams that do this)*	Django People / Core Team	Since one of the responsibilities of the core team is to develop, review and merge patches, they also represent the developers among the rest of the community

Type & Description	Stakeholder	Observations
Maintainers *Manage the evolution of the system once it is operational*	Technical Board / Ops Team	The technical board goal is to maintain the project, while the Ops Team maintains the infrastructure
Suppliers *Build and/or supply the hardware, software, or infrastructure on which the system will run*	Operating Systems / Python / Javascript	Django runs on MacOS, Linux and Windows. Besides a host, Django also needs Python and Javascript to run, giving the fact that is a Python web framework.
Support Staff *Provide support to users for the product or system when it is running*	Core Members / Django People	The Core Members have the authority over the IRC channel, Issue tracker, mailing list, where other community members may contribute too. Moreover, all the people that are engaged into conversations on StackOverflow are part of the support staff
System administrators *Run the system once it has been deployed*	Ops Team / Releasers	The releasers are concerned with releasing the project to the public, while the ops team maintains the servers
Testers *Test the system to ensure that it is suitable for use*	Developers	Django follows the Test Driven Development methodology, therefore the developers are also responsible for writing test cases and running the tests
Users *Define the system's functionality and ultimately make use of it*	Community Members / End Users / Content Creators	The members of the community are part of the users, because they use Django and also create Trac issues to help improve the project or create new functionality. The end users represent the developers from companies such as Instagram, Spotify, Mozilla FireFox, Bitbucket and Disqus, who are using Django to develop their products. The content creators are the people who are using Django to help other users by publishing blog posts, tutorials and videos.

6.3.4   Other stakeholders

Besides the clasification introduced by Rozanski and Woods, we have identified other types of stakeholders that influence the development of the project from other perspectives.

Competitors The competitors are one of the stakeholders that influence the project from a different point of view: competition. Based on what other frameworks can or can not do, the project might adjust its own capabilities. Therefore, the main competitors identified are either other major Python web frameworks, such as Flask or Pyramid, which are more lightweight but lack some functionalities, or fully-fledged web frameworks, such as PHP's CodeIgniter.

Translators Translators are the people who help with the project internationalization. The framework has a very well defined methodology to help translators, described in developement view chapter.

Founders The Django founders are Adrian Holovaty (web-developer, journalist and antreprenor) and Simon Willison (programmmer, director of architecture Eventbrite).

Donators Django accepts donations that are used to support, promote and advance the framework. The donors might be individual persons or companies, who can be found at the end of the fundraising page. Their help further support programs such as Django Girls, the fellowship program or events and conferences in general. The donators can contribute using one of the several methods for donations such as: + Django's own donation page where one can choose the type of donation (monthly, quarterly, yearly or one-time) + mailing checks to the foundation address + payroll deduction via Benevity Workplace Giving Program + through Amazon Smile.

6.3.5   Power-interest Grid

Django has a very interesting stakeholder structure, the majority of functions being occupied by the people belonging to the Django Software Foundation together with the Core Members and Technical Board. Therefore, the trio has the highest interest and the highest power. Secondly in terms of power are the Corporate members, which have a lower interest than the previous, but high enough, because they care about the future of the project. Next in power are the suppliers, which have no interest, but the project has a high dependency on them and each change in the suppliers affects imediately the project. The donators have the same power, because they care enough to donate for the project, having the power to support many aspects such as conferences, events and programs.

Furthermore, there are the entities who do not have a high power, but have a high interest in the project. These are the users, the active developers and the founders who still care about the project, but are not as involved anymore. Next, with a very high interest are the competitors, which have a limited influence over the project.

Finally the less active community and the Stackoverflow community have little power in the project development, but their impact is very high for developers and users that struggle with the framework and seek guidance.
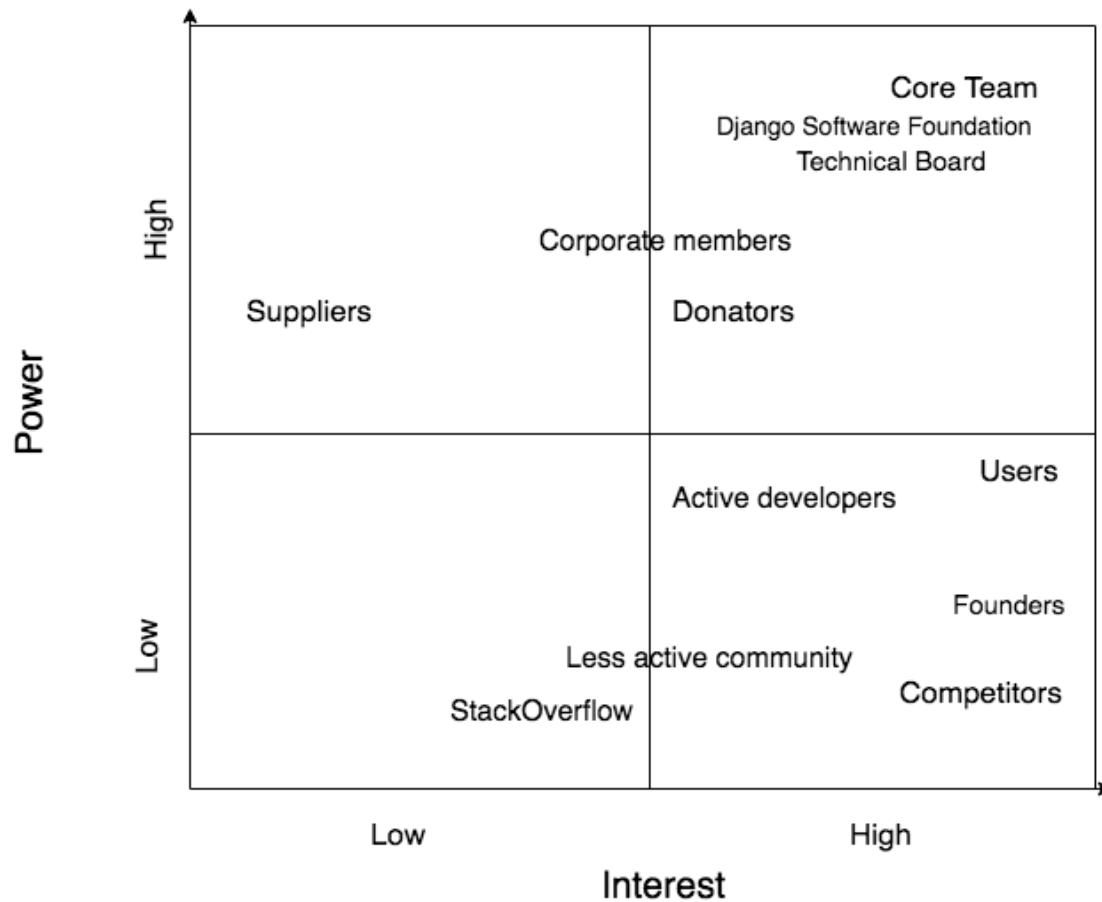
Figure 6.1: Power-interest grid

6.4 Decision-making process of pull requests

After careful analysis of 20 pull request, 10 accepted and 10 rejected ones, we were able to gain a thorough insight into decision-making process in the Django project organization.

All pull requests must relate to a ticket, which describes a bug or a new feature, from the change management tool Trac which is publicly accessible on the Django project website. The ticket has to be assigned to the the author of the pull request on GitHub. This process does not apply to security bugs which are handled by members of the Django project team themselves.

After a pull request is submitted, it is usually first discussed amongst and tested by other contributors. The author usually makes some changes based upon that feedback and if he thinks the quality of the pull request is sufficient, he tags a team member of the Django project team to review it.

The integrators from core Django team are very strict about software quality and all pull request are carefully analyzed. Apart from pure functionality, this also includes aspects like code formatting & style, proper documentation and test coverage. Regression tests are run against each pull request which even include performance tests. They often demand changes and make suggestions on how to implement them. From that point on the team member stays actively involved by repeatedly assessing the patch until either all the quality standards are met and the patch is merged or the author fails to meet the demands of the integrator and the patch is rejected.

All integrators seem to work independently and can alone decide if a pull request should be merged or rejected. More complex or controversial changes are sometimes discussed by multiple members of the integration team or over the users email-list.

In conclusion, Django has very well established and regulated decision process that is strictly followed and is designed to keep the quality of the project on its high level.

6.5 Context view

In order to truly understand the architecture behind Django, one needs to first understand the context surrounding the project, its scope, responsibilities and dependencies.

6.5.1 System Scope and Responsibilities

Since Django is a web framework, the scope of the project is restricted to building web applications only and since it is a full stack framework, it is suitable for various use cases, such as social networks, internet banking applications etc. Therefore, we have divided the main responsibility of allowing the building of a web application in multiple sub-responsibilities:

- Allow users to render data on the screen or fetch data (for example, through inputs).
- Allow users to create re-usable page templates.
- Allow users to build a server infrastructure that can receive requests and return appropriate responses.
- Allow users to build encapsulated modules that can function separately from others, communicate and maintain their own state.
- Provide means and specialized components to connect and read/write from/to persistent storage.

- Provide pre-defined modules and functionality that solve common Web-development problems.
- Provide specialized modules that ensure the security of the web application.
- Provide the flexibility of interfacing to other libraries of frameworks.

6.5.2   Context view diagram
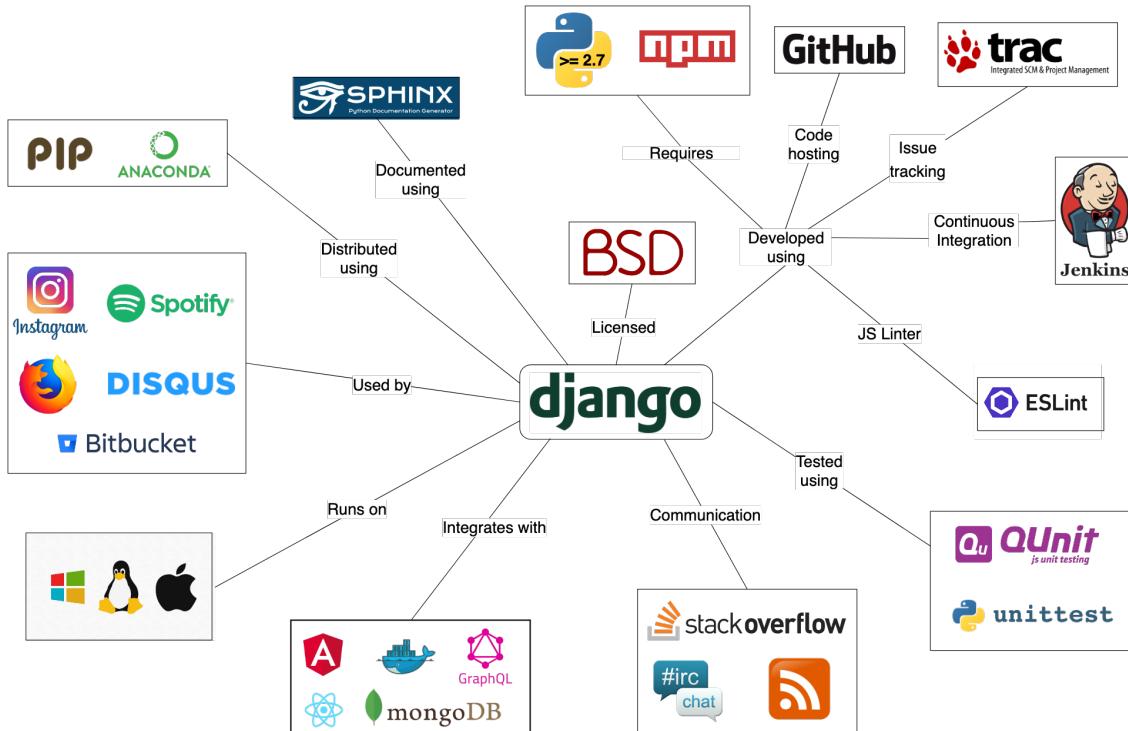
Figure 6.2: Context view

The figure above shows a context model implementation for the Django project. By analyzing it, we were able to uncover several relevant findings:

- Django **runs on** all major desktop-grade operating systems (Windows, MacOS, Linux, Unix)
- Django is **used by** important players in the tech industry, such as Instagram, Spotify or Bitbucket.
- The team uses a variety of **development tools** inside the project: GitHub for hosting and collaborating on the code, Trac for issue tracking and Jenkins for continuous integration. Moreover, in order to fully run the entire project, both Python (>= 2.7) and npm are **required**. New functionalities are **documented** in .txt files and generated using Sphinx
- The project is **tested** using Python's built-in unittest framework and QUnit for Javascript code. Other testing frameworks can be used, as Django provides an API and tools for this type of integration.
- Django can be **integrated with** various other frameworks:
  - Front-end web frameworks, such as Angular or ReactJS
  - Various types of databases, such as GraphQL or MongoDB
  - Virtualization solutions, such as Docker

- You can discover extensive **discussions** about the framework on StackOverflow and you can stay updated regarding the latest developments either through the IRC Channel or the RSS feed. More details about getting in contact with the Django community can be found here

6.5.3   Conclusion

To sum up, by modelling the context view, we have discovered that Django has various responsibilities when it comes to functionality. Moreover, we have found there are many dependencies surrounding the projects.

6.6   Development view

6.6.1   Module structure

Django is organized in multiple modules that encapsulate different functionalities of the project and serve different purposes. We have identified 3 high-level modules, which are then organized into multiple packages. An interesting remark here is that we have noticed a similar structure in DESOSA 2016 for Ruby on Rails (although less complex and with less modules). When investigating the release page of Ruby on Rails, we have discovered many references to Django, which shows that the architecture of the 2 is very similar.

6.6.1.1   Main Django module

The main Django module, which is encapsulated in the /django folder, contains the actual functionality of the project. It is composed by various packages, that contain mostly Python scripts, but also Javascript, CSS and HTML. However, in order to better understand how various functionalities are grouped inside this module, we have also evaluated the grouping of the sub-modules. We have identified 8 groupings that reflect different functionalities that the project offers.

Core functionality & configuration This grouping handles the application configuration and low-level functionality and contains 2 packages:

- The conf package contains the default global configuration settings of the application, such as environment variables, locale formats etc.
- The core package, contains the core low-level functionalities that the application needs, such as caching, exceptions, serializers, file operations.

The functionality in these packages is used in various places in the application, especially in the lower levels.

Communication Django has a built in signal dispatcher, which helps allow decoupled applications to receive notifications when a specific action occurs. The dispatch package contains the implementation of this functionality, which is used by various other levels for event signaling.

Persistent storage functions As most application nowadays require some form of persistent storage, Django incorporates a db package, which acts as an ORM for a variety of databases.

State change handlers As Django is a web framework, it uses HTTP request and response objects to pass state through the system. There are 2 packages that encapsulate this functionality:
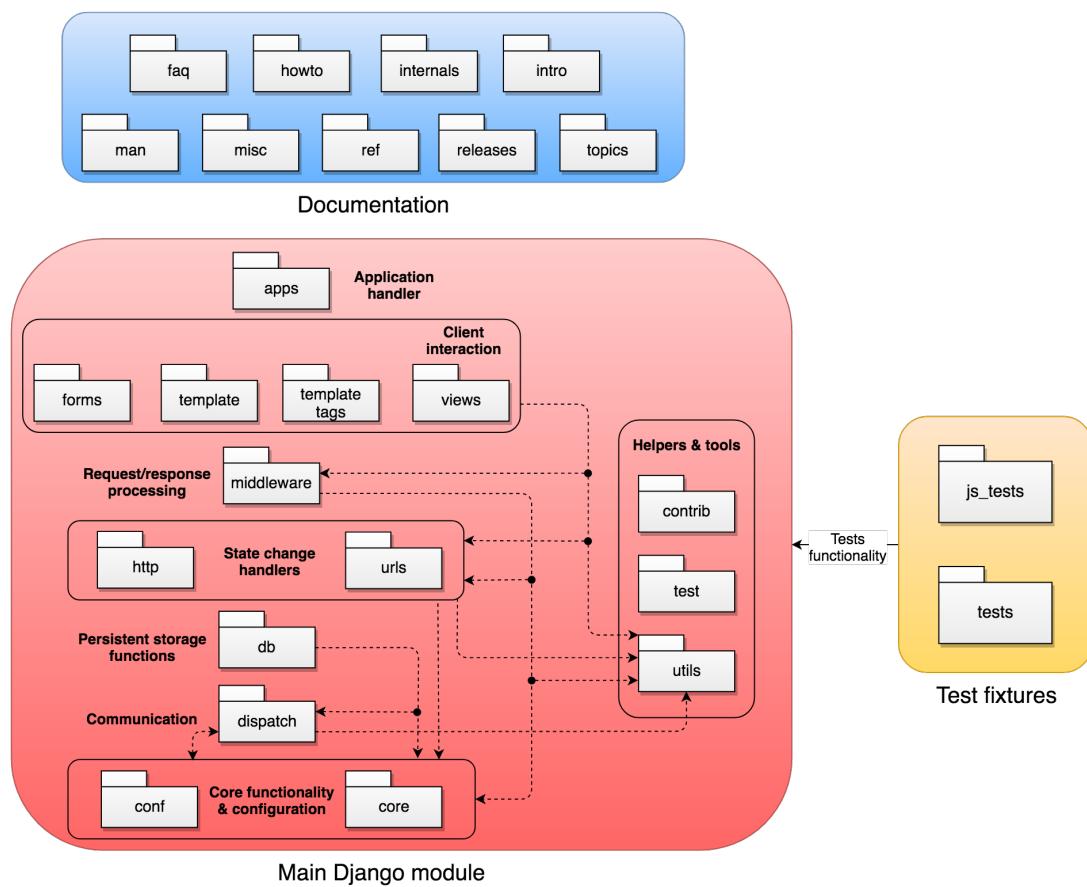
Figure 6.3: Django modules

- The `http` package implements the Response and Request objects and all the needed functionality (encoding, scheming, headers, cookie parsers etc.).
- The `url` package handles parsing and processing of URLs in order to do matching with various components, such as a view.

Request / Response processing The `middleware` package is the intermediary between the front-end architecture and the server. It handles tasks such as communicating with the cache, processing requests and responses, enforcing security policies etc. This package is mostly used by the client interaction packages.

Client interaction This grouping encapsulates the whole front-end architecture and can be described by 4 packages:

- The `views` package contains the implementation of the most important component of Django's frontend architecture: a view. Every Python function that receives a web request and returns a response (e.g. web page, document, JSON etc.) is considered a view.
- The `forms` package contains various functions that help implement and operate forms (render forms, fetch inputs etc.).
- The `template` package contains the implementation of Django's template system, which enables the developer to re-use various web page components or custom elements. The tags and filters that are used by the template system are defined in the `templatetags` package.

Helpers & Tools The framework has 3 packages that encapsulate various helper functions and utilities that can be used throughout the different levels of the architecture:

- The `contrib` package contains various optional tools that solve common Web development problems, such as a search functionality with query autocompletes, an admin page, sitemaps etc.
- The `test` package contains utilities for building tests quicker and easier
- The `utils` package contains various helpers for aiding development in different parts of the application. It is one of the most used packages throughout the rest of the framework.

Test fixtures The Django team uses *test-driven development* as a programming paradigm. This means every new functionality or bug fix needs a corresponding set of tests. The tests cand be found in 2 packages, depending on the programming language the tests are made for. Therefore, the Python tests are stored in the `/tests` folder, while the Javascripts tests are stored in `/js_tests`.

Documentation module Since Django is a full stack framework, that means the documentation can also be stored in the project and served on the main project website. The documentation module, which can be found in the `/docs` folder of the framework, contains the whole documentation, written in .txt files. It contains pages such as basic tutorials (`intro`), the API Reference (`ref`), in-depth how-to guides (`howto`) or releases (`releases`).

6.6.2   Common patterns

In Django, there are multiple patterns used, each being applied to a certain piece such as the models that follow the "Active record" design pattern or the templates that are designed to follow the principle of "template inheritance", therefore avoiding duplicate code. Moreover, Django is object oriented and the development follows the test-driven development process.

Django design philosophies As the official documentation of Django explains, they have included several principles or patterns during the project's life. Overall, they aim for: * loose coupling and tight cohesion,

where the layers of the project should not depend one on another unless is necessary; * less code - which aims for using as much as possible the Pyhton's capabilities, such as introspection. The downside is the high dependency of Django on Python, which requires careful maintanance in order to comply with Python's changes. * quick development - which is the most wanted feature of a framework from the user's point of view. * adhering to the "Don't Repeat Yourself" Principle - therefore, each component of the application should be in one and single place, which increases the normalization and reduces the redundancy. * consistency - by leveraging Python * explicit behaviour - which means that the code should not assume certain aspects, such as properties or data types of the models.

6.6.3 Common processing

Message logging In Django, message logging is being done using Python's built in logging module.

The message is forwarded from the logger to the *handler*, which decides the destination of a message. *Filters* can be configured to have additional control over which records are passed from the logger to the handler.

Testing The testing process can be observed in the figure below and is split by programming language and level of detail.

Documentation Documentation has an important role in the project and the members aim to improve it whenever is necessary. The documentation uses Sphinx system which is based on docutils, which transforms plain text into pdf or other prefered output format. The language is reStructuredText, which is a plain markdown language used in both Docutils and Sphinx. Besides the Sphinx markup, Django introduced an additional set of tags such as: *setting, templatetag, templatefilter, fieldlookup, django_admin, django_admin_option and ticket*. To compress documentation images, the project uses OptiPNG and AdvanceCOMP. Moreover, the spell check is done using additional tools such as pyenchant and sphinxcontrib-spelling.

6.6.4 Standard design

Coding style Django Project contains besides the Python code, Javascript code to develop the front-end modules *admin* and *gis*. Therefore, the project follows two different coding style conventions for each programming language.

The Python style follows the PEP-8 Python style guide. Moreover, Django sets additional rules such as: the line length, identation, alignment, usage of single or double quotes. They also standardised the variable name notations and imports style, which uses isort library to sort the imports according to predefined rules. Another helper used in the project is flake8, a style checker that identifies mistakes.

The Javascript style follows the general Javascript conventions. The main difference between the two programming languages is the naming notation which is **camelCase** in Javascript and **underscore_case** in Python. The project uses JSHint code linter to analyse the code for mistake in style and other bugs. Moreover, to compress the Javascript code, the project uses closure compiler developed by Google.

To ensure the developers use the same identation style, the project contains the *.editorconfig* file for both Python and Javascript.

Internationalization and localization Django supports internationalization and localization for text translation, time zones and formatting dates, times and numbers.

runtests.py

- Simple tests

Basic unit testing

- No special configuration

- -

- Multiple virtual environments
- Import sorting

Extensive unit testing
- Code style
- Documentation

- -

QUnit *js unit testing*

Javascript testing
- Tests code for *admin* and *gis* contrib apps

- -
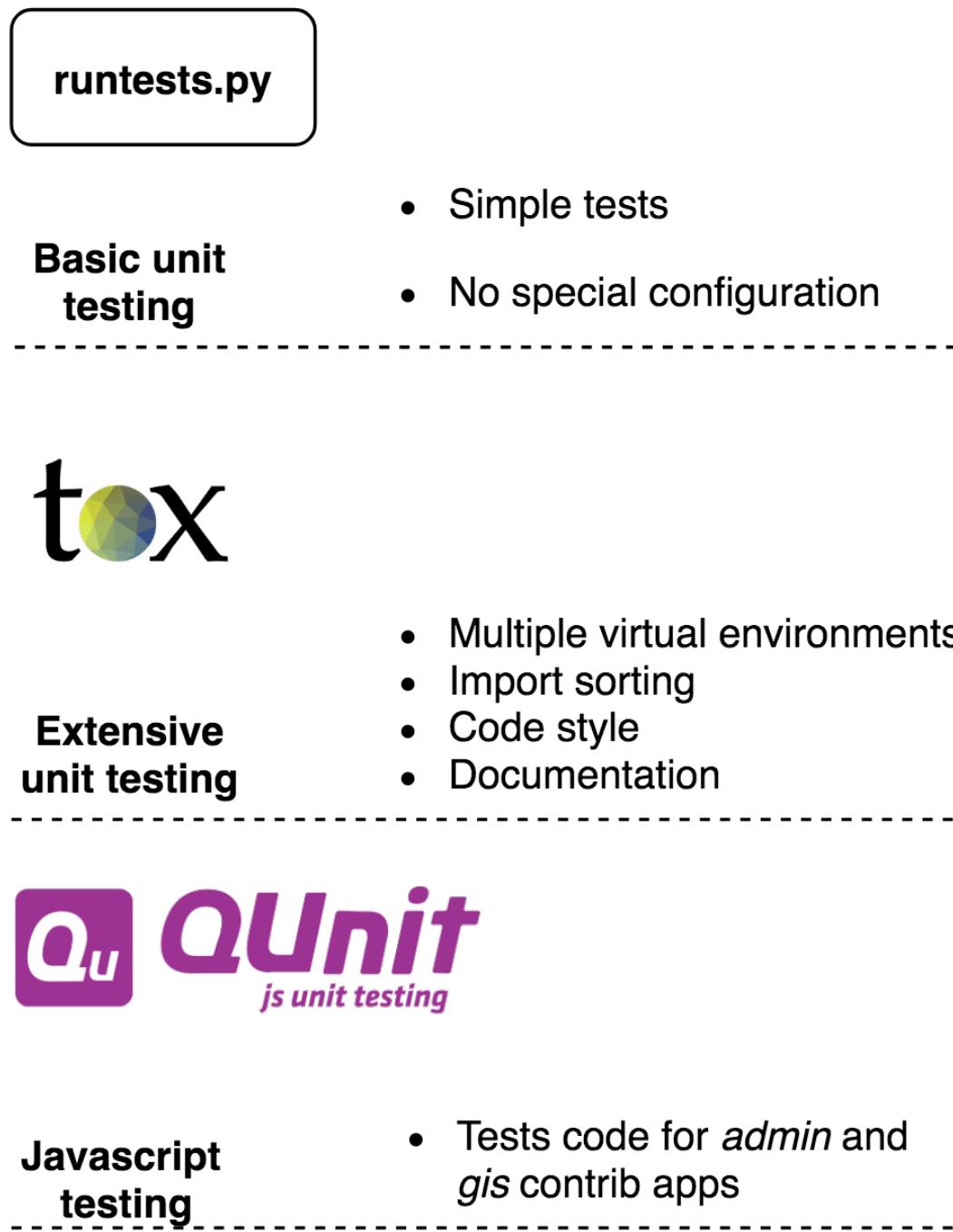
Figure 6.4: The testing process

To perform translation, Django uses "translation strings" which represents strings that need translation. Then, the language dependent corresponding string is transformed using GNU gettext toolset, which retrieves the text from *.po* files. Moreover, Django supports plural words, by extending *gettext*. The Javascript translation raised a few challenges, because it could not access *gettext* by default. Therefore, Django made its own implementation to address translation in Javascript, called *JavaScriptCatalog view*.

Moreover, to monitor the translation activity, Django uses Transifex for both the code and documentation.

6.6.5 Standard software components

Javascript libraries The additional libraries for Javascript used are related to code inspection and project building, which are all included in the package.json file.
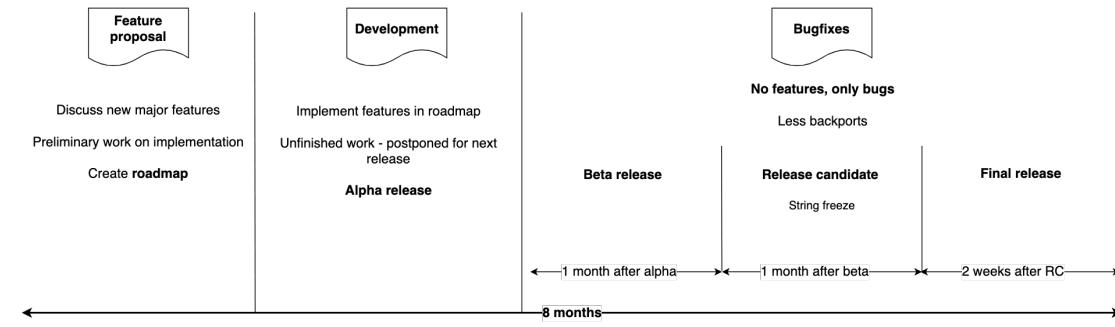
Python modules Django uses additional Python modules only for running the test suite. The external modules installation represents a requirement for test running, which can be found in the requirements folder under tests.

6.6.6 Build approach

Every developer is requested to submit a pull request (PR) which is reviewed by core developers. For each PR, 19 checkes are performed using Jenkins. These checkes perform several builds using different environments to properly test the framework. The builds test the documentation for spelling mistakes, the code style, the imports, runs the Javascript tests, runs the project on Windows, on Ubuntu, on Selenium, on Oracle and use different databases to check the compatibility.

6.6.7 Release process

The Django team schedules a major feature release every 8 months. Throughout the period, 3 major phases can be identified: feature proposal, development and bugfixes. The timeline, along with the characteristics of each phase, can be observed in the figure below. The final steps of the lengthy process are detailed in the pre-release and release checklists in the figures below. The pre-release process starts approximately one week before the actual release.

The release timeline

Pre-release checklist

✔ Check for uncomitted code

✔ Check Trac for blockers

✔ Go through the release notes

✔ Go through existing Pull Requests

✔ Check translations of new features

✔ Update admin page

Release checklist

✔ Check Jenkins output

✔ Remove "Under development" headers

✔ Update release date

✔ Update version number & git tag

✔ Generate release package

✔ Sign checksum & publish

✔ Announce on blog

Figure 6.5: The release checklist

6.7 Technical debt

Technical debt represents the costs of choosing a "quick and ugly" solution now, instead of using a proper approach that is harder and slower. We present the results of our code and test analysis in this section to assess Django's debt.

6.7.1 Code quality tool analysis

We used SonarQube to gather various code metrics in order to find technical debt in Django. This tool provides information about application health, code quality, bugs and many more. We chose this tool, because it analyzes code written in python, JavaScript and HTML, which are all used in Django. We analyzed the current version, as well as 2 previous versions, to see how the technical debt changes over time.

6.7.1.1 Scan results

Version	Bugs	Vulnerabilities	Code smells	Duplicity	Lines of Code
2.2	73	189	1193	1 %	263 393
2.1	73	185	1165	1 %	255 251
2.0	66	185	1152	1 %	251 107

6.7.1.2 Bugs

The current version has 73 bugs. Considering Django's 263 000 LOC, 1 bug per 3600 LOC does not represent a major issue.

Taking a closer look at the bugs, we observed that most of them are several years old. Therefore, it is unlikely that nobody noticed them for such long time, if they can be easily detected with automatic tool. It is more likely that these bugs are not important enough to fix them or they do not want to edit components that have been closed for several years, for minor bugs that are not causing any significant problems.

Bugs are mainly consisting of wrong or missing HTML tags in the documentation section, potentially undefined variables and too many relational operators in if statement. They are also present in the older versions.

6.7.1.3 Vulnerabilities

SonarQube reports 173 vulnerabilities. They are composed almost entirely of statically set IP addresses in the test suite, which does not represent a major problem because all the tests are for the internal use of Django. These vulnerabilities are present in all tested versions.

6.7.1.4   Code Smell

Code smell refers to any symptom in the source code that may indicate a deeper problem, although they are not technically incorrect and do not currently prevent the program from functioning. They indicate weaknesses in design that may slow down development or increase the risk of bugs in the future.

Code smell type	Occurences
Incorrect name of variables	553
Functions with high cyclomatic complexity	380
Non-standard format of comments	135
Code leftovers for legacy support	119
Potential if merges	59

In Django there are less than 1200 code smells. When we compare different version of Django, the number of code smells stays practically the same. That is an indication of improvement of quality control in the last years, because most of the issues are the same across the versions. A closer analysis of code smells can be seen in the table above. SonarQube estimates that solving all the code smells will require approximately 528 hours of work.

6.7.1.5   Duplicity

Duplicity of code in Django is about 1%, but 87% of duplicated lines are in tests, which is understandable. Moreover, the number of duplicities is decreasing with newer versions.

6.7.1.6   Ticket System

Django project uses the Trac ticket system for bugs and optimizations, of which there are currently 872 open tickets. The Django team should consider to open tickets to get rid of the technical debt presented here, since this needs to be handled proactively.

6.7.2   Testing Debt

Testing is an important tool to asses and control the quality of a piece of software. Therefore a weak test suite or low test code coverage are part of a systems technical debt. Especially for a complex project like Django, which offers a significant amount of functionality and has a big user base, maintaining an extensive test suite is of paramount importance.

6.7.2.1   Testing practices

The test suite contains both testcases for the Python and the Javascript code. They are implemented using the unittest python module and QUnit respectively. The central piece to executing the tests is the *runtests.py* script. This is usually run through the tox build tool. The tool can be executed using different python

versions, which assures backwards capability of the test suite. This prevents technical debt since it decouples the runtime from the tests themselves.

Since Django is a full stack web framework, the execution of the test suites depends on the database management system that is used in the stack. Django supports Oracle, PostgreSQL, MySQL and SQLite by default. Upon invoking tox, a setting file can be passed as an argument, that specifies which DBMS to use and how it should be set up. The test suite comes with a setting file for SQLite as the default system, since it is usually part of the Python installation. This also prevents technical debt since it makes the data layer of the stack completely interchangeable.

6.7.2.2 Test coverage

The Django development team has done an admirable job and the test suite reaches a 79% line coverage and an 77% branch coverage when run with the standard settings. This is achieved mainly through the strict guidelines, that all pull request have to adhere to which enforce the extension or adaption of the test suite, for any change that is made to the code.

When looking at the roughly 20% of the code that is not covered by the tests, they can quickly be identified as modules that work with another database management system, for example PostgreSQL. If the test suite is run using the PostgreSQL settings, it reaches 81 % line coverage and 79% branch coverage. Again the uncovered modules are the ones depending on a different DBMS.

When calculating the test coverage over all possible database backends, the test suite reaches a 97% line coverage and a 95 % branch coverage.

A detailed inspection shows that the few lines and branches that are not covered by the tests fall into three categories: throw exception statements, default return value statements (such as empty string) and error log statements. What these all have in common is that they do not test the happy path of program execution. Extending the testing guidelines to enforce the coverage of error branches in the program execution would help to get rid of the little technical test debt the project still has.

6.7.3 Conclusion

Based on our analysis we can state that Django has very little technical debt. Given the size and complexity of the framework, this is very impressive. If the code quality standards keep being enforced so rigorously, the development of Django should not run into any future problems because of technical debt.

6.8 Evolution Perspective

This section analyses the evolution and history of Django. The evolution perspective focuses on how the software evolved throughout the years and how it dealt with changes and problems. According to Rozanski and Woods, a software system should be flexible enough to deal with all possible types of changes, that it faces during its lifetime. We identify and discuss major changes to the Django project over the years and analyze their causes and consequences.

6.8.1   History

The Django project was started in the autumn of 2003 by Adrian Holovaty and Simon Willis. They both worked for the Lawrence Journal-World newspaper and initially tried to build a content management system in Python to publish news articles. They discovered how easy and fast it was to create web applications with python and therefore expanded their vision for the project to build a full stack web framework. They worked on it in private for 2 full years and as there is now public record of that time, it is impossible to discuss the evolution of the project in that stage.

They released the first public version of Django (0.9) on 16 November 2005 under the BSD license. Even though the developers clearly stated that it was still a work in progress and discouraged its use in commercial systems, the project already had an astounding level of technical maturity. It had a test suite with moderate coverage, comprehensive documentation including tutorials and multiple communication channels for the community to follow and contribute to the project. Furthermore it came with its own lightweight webserver and supported multiple popular database management systems. That Django never had to go through a major rewrite throughout its live time can be partially attributed to the fact that its evolution started from a very solid base.

The project quickly attracted a lot of interest from web developers who were tired of the complex and heavy weight LAMP stack that was prevalent at the time and a very active community started to grow around the project. This community has been one of the driving forces behind the evolution of Django and helped to improve and expand the framework with every release.

6.8.2   Aspects of evolution

6.8.2.1   Release cycle

At the start of the project, releases were not planned based on regular time frames. While they did communicate release dates to the community, these were often postponed and they followed more of a "it's done when it's done" philosophy. They also had a bus factor of 1 at this point, as only one core member was responsible for publishing the releases. As the team gathered experience over the years, the delays became fewer although the time between major releases was usually between 6 and 12 months. In 2015 they conducted a survey amongst contributors and release a roadmap based upon its results, where they vowed to release a major version every 8 months and a long-term support version every 2 years. Their minor releases however do not follow any policy or roadmap. Minor releases have always been rolled out when needed, usually to fix a number of serious enough bugs or to patch security vulnerabilities. This has been an important tool to deal with sudden changes through the evolution of Django.

6.8.2.2   Security issues

Since projects built with Django usually get exposed on the internet, security issues have been the most serious threats to Django's evolution. The way of patching these vulnerabilities is strictly handled internally, even though Django leans heavily on its open source community for normal bugfixes. Security has always been treated as a first-class feature by the Django team and they reliably fix any issue in a short amount of time and roll out a minor release as fast as possible. Thanks to this policy, Django never had to deal with a major security scandal in its evolution.
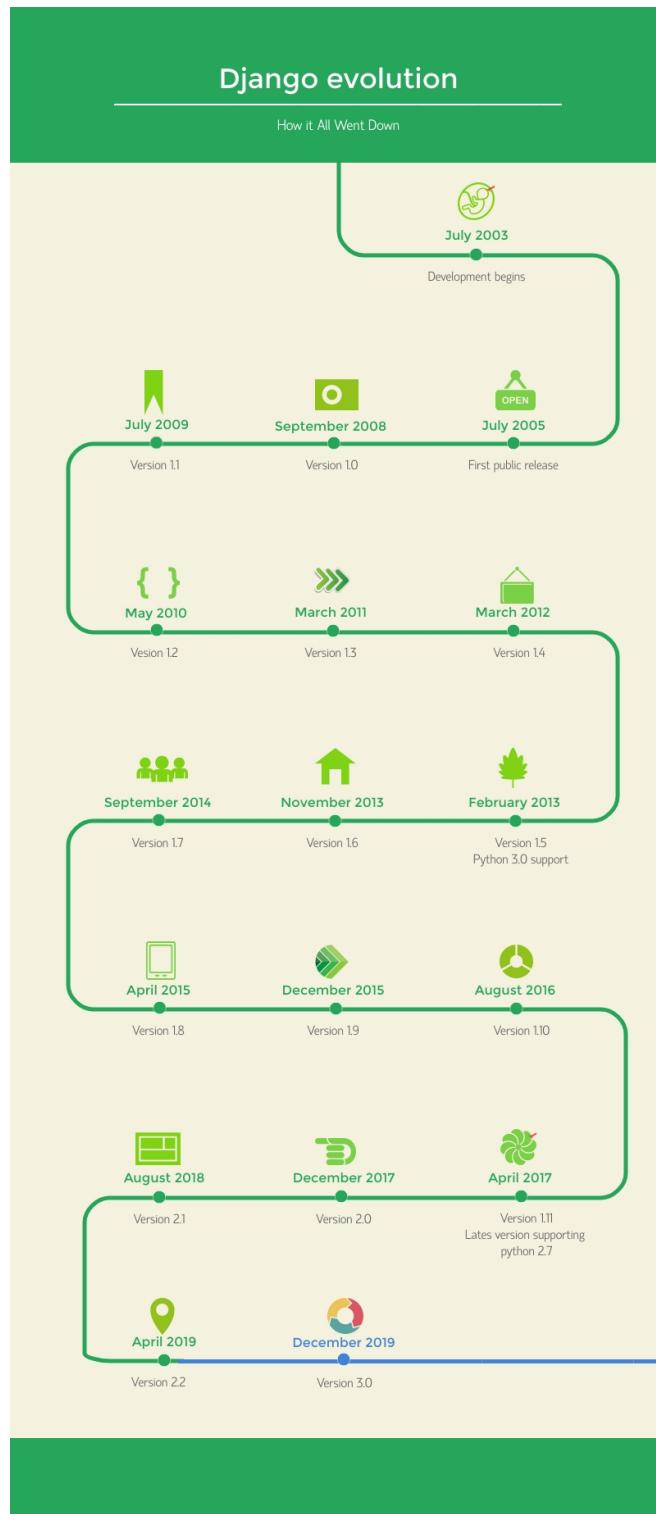
Figure 6.6: Django timeline

6.8.2.3   Refactoring

The database API of the initial release (0.9) was criticized by the community for begin hard to understand and containing to much 'magic' code. This prompted the first refactoring of a module in Django's evolution. Having an easy to understand codebase has been part of Django's design philosophy from the start and throughout its lifetime various modules have been subject to refactoring as part of major releases.

6.8.2.4   Deprecation management

Backwards-compatibility is of major concern to the Django developers and their policy since the first stable release 1.0 has been to avoid introducing any changes that would make a migration of project to a newer Django version difficult. A notable exception to this are changes needed to patch security vulnerabilities. However as the project evolved and matured, the need for certain features and mechanics to change and modernize arose. The developers therefore came up with a two phase deprecation process: features can become deprecated in a major release and will be removed in the following major release. Minor releases do not have any impact on this process. This gives the community enough time to change parts of their project that uses deprecated features and assures that the Django code base doesn't get cluttered with functionality nobody uses anymore.

6.8.2.5   Dependencies

The Django team is very aware of changes in its ecosystem and tries to keep up with them by incorporating updates in their major releases. A showcase for this is its strongest dependency on Python, the programming language it is implemented in. In the release 1.2, they updated the required Python version for the first time from 2.3 to 2.4. They have kept up with Python's development, as well as other dependencies such as the various database management system Django supports, throughout the framework's evolution, as the current 2.2 release requires at least python 3.5.

6.8.3   Evolution conclusion

To summarize, the evolution of Django has been a well-managed process, guided by smart decision of the core developer team and supported by its active community.

6.9   Conclusion

Django is a highly maintained project, with a big community that helps the project growing by suggesting possible features, reporting bugs and also contributing to the code. Thus, the repository has daily commits, code reviews and merged pull requests.

Since the project started in 2003, we have found the architecture to be very mature. Given the size of the codebase, we have found very few bugs and code smells, showing that the code maintenance is performed thoroughly. Moreover, we have found that the high quality of the code is a consequence of an established set of code reviewing guidelines, which is always respected by the core developers.

Chapter 7

Flair

7.1 Table of contents

121

7.2   Introduction

Flair is a state-of-the-art open source natural language processing framework developed by Zalando Research. Zalando Research is part of the fashion platform Zalando that aims to scale technology in fashion with the power of experimentation in theory and in practice. Flair released in July 2018 and is a novel approach where natural language modeling is leveraged to learn powerful, contextualized representations of human language from large corpora. Such representations possess a multitude of semantic and syntactic information that can be used to directly improve downstream NLP tasks.

The standout features of this powerful NLP library lies in the usage of Named Entity Recognition (NER), part-of-speech (PoS) tagging classification and sense disambiguation. It boasts support for a variety of languages and continues to grow with the help of many contributors. It consists of taggers that are 'one model, many languages', that is, a single model can predict the PoS or NER tags for given input text in various languages. Flair also provides a text embedding library with simple interfaces that allow for usage and combination of different word and document embeddings including the proposed Flair embeddings, BERT and ELMo embeddings. It is built directly on PyTorch which allows for ease of use in training and experimenting with models.

7.3   Stakeholders

Stakeholders in a software system refers to the individual, team or organisation that have interest in said system being realized. Each of these stakeholders have their vested interests and needs from the system. A myriad of stakeholders are involved in activites such as building, testing, debugging, etc. It is imperative to identify individuals or groups that are likely to affect or be affected by a proposed action and sorting them according to their interactions. Stakeholder analysis is done so as to identify and prioritize the role of the stakeholders.

We use the description of stakeholders obtained from Rozanski and Woods [1] to classify stakeholders. These stakeholders will be mentioned as **Main Stakeholders**. Stakeholders not part of the Rozanski and Woods classification, are described as **Other Stakeholders**.

7.3.1   Main stakeholders

---

[1]Rozanski, N. & Woods, E. (2011). Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives. Addison-Wesley.

Type	Description
Acquirers	In Flair, control is orchestrated by Zalando which works on several real world problems. The researchers work on problems with the academic community, synergising with company objectives. Achievements of Flair will allow Zalando to properly utilize their resources.
Assessors	The conformance of Flair to research standards is regulated by Zalando. The quality assurance is divided within contributors at the GitHub repository. Alan Akbik, who wrote Contextal String Embeddings for Sequence Labeling with Duncan Blythe and Roland Vollgraf, mainly oversees the commits along with Dr. Kashif Rasul and Tanja Bergmann.
Communicators	Communicators act as conveyers of system specifications. To follow updates of Flair, watching the repository on GitHub is one way, followed by updates on their website. Akbik, Bergmann and Rasul are the communicators, which allows them to understand insights provided by other contributors.
Developers	Initial attribution goes to the research team for their work on the paper. The other contributions come from developers and researchers. This allows the original developers to gauge the performance of the framework and identify areas of potential growth. Akbik has the most commits with 445, followed by Bergmann with 276, and Rasul with 31. GitHub users such as Stefan Schweter are active during development. There are 36 contributors.

Type	Description
Maintainers	Bug fixes and maintenance are handled by both the developers and GitHub contributors. The maintainers are Akbik and Bergmann. They maintain the development of the library and check for functionality and version control.
Production Engineers	Potential Production Engineers, would want to manage releases and run tests on builds. At the moment, the task is relegated to staff developers. The focus should also be on ensuring that the library performs efficiently. The deployment of Flair is the responsibility of users.
Suppliers	The infrastructure used is that from the user's side and therefore the users in a way are also suppliers of hardware in addition to Zalando. GitHub is also a supplier as it provides a common platform for contributors to develop. Google Colab provides alternatives for hardware constraints.
Support Staff	The Zalando Research team acts as support staff. Per Ploug acts as an intermediary between the external contributors and Flair developers. This also includes Henning Jacobs, the Head of Developer Productivity at Zalando SE. Her team uses various tools to measure productivity of the developers.

Type	Description
System Administrators	Since the library is used by users, they become system administrators themselves. This is also overseen by the Zalando team. System administrators are given the role of identifying issues with data associated with labeling and embeddings, performance problems and deviations from the desired non-functional performance requirements.
Testers	Testers in Flair are the core developers and other members of the research team. This includes Akbik, Rasul, Bergmann, Nikolay Jetchev and external testers include GitHub users such as Schweter, David Batista, and Nilabhra Chowdhury.
Users	The functionality of the system is used by students, researchers and companies. The scope of the project is understood mainly by people already well versed in NLP. Flair is already part of several in-production systems at Zalando, as machine learning has become a natural part of their engineering toolbox.

Table 2.1: Main stakeholders of Flair

7.3.2 Other stakeholders

Researchers and Scientists: They use the system to research and present findings. The use of embeddings and labelling in tagging are of vital interest.

Competitors: The most widely known library is NLTK which supports tasks like classification, tokenization, stemming and semantic reasoning. It is difficult to use and considerably slow. Gensim is a Python library that specializes in identifying semantic similarities between two documents through vector space modeling.

Translators: At the moment, the NLP tasks can be performed on languages such as English, Dutch, Spanish, German, Polish and a few others. Other languages are not supported yet and Flair invites developers to contribute to the corpora and the embeddings list in order to support more languages

Figure 7.1: Flair's Stakeholders

Enthusiasts: There are several students and developers who are interested in performing NLP tasks and would therefore find contributing to Flair an exciting way to broaden their horizons.

Bloggers/Social Media: They are unambigous and provide reliable sources of information for new developers to start contributing. They also help bring the project into limelight. This article is an example of a website providing a tutorial and this article is an example of a blogger sharing their insights.

Sponsors: Even though the library is an open source, it originated from the Research Team at Zalando. Setting up of architecture and funding for research is provided by Zalando. The results of the research are essential for Zalando to further its business operations.

Integrators: Integrators are the architects and core developers of Flair who are responsible for maintaining code standard and consistency of Flair. Developers like Akbik and Bergmann are the main integrators here. To keep the framework stable, they merge Pull Requests (PR) which they successfully align with their versions of sources and languages. When a new module is added, they check if the change is needed and also verify documentation changes. This procedure helps maintain the quality of the framework, keeping it open source as well. To discuss the challenges of the integrators, one important aspect noticed is that sometimes, a major part of the PR works on test set whereas a small part does not give expected test result. In these cases, the PR is not directly closed. It is kept as open, the integrators comment on the errors they face, and suggest what changes can be created to overcome that. Sometimes, the same contributor makes the required changes or other contributors have a chance to solve the issue as well. After reviewing the code and checking for errors, the PR that meet their requirements are merged.

7.3.3 Stakeholder analysis: Power vs Interest Grid

Based on Mendelow's Power vs Interest Grid [2], the groups of stakeholders necessary to be managed closely are classified.
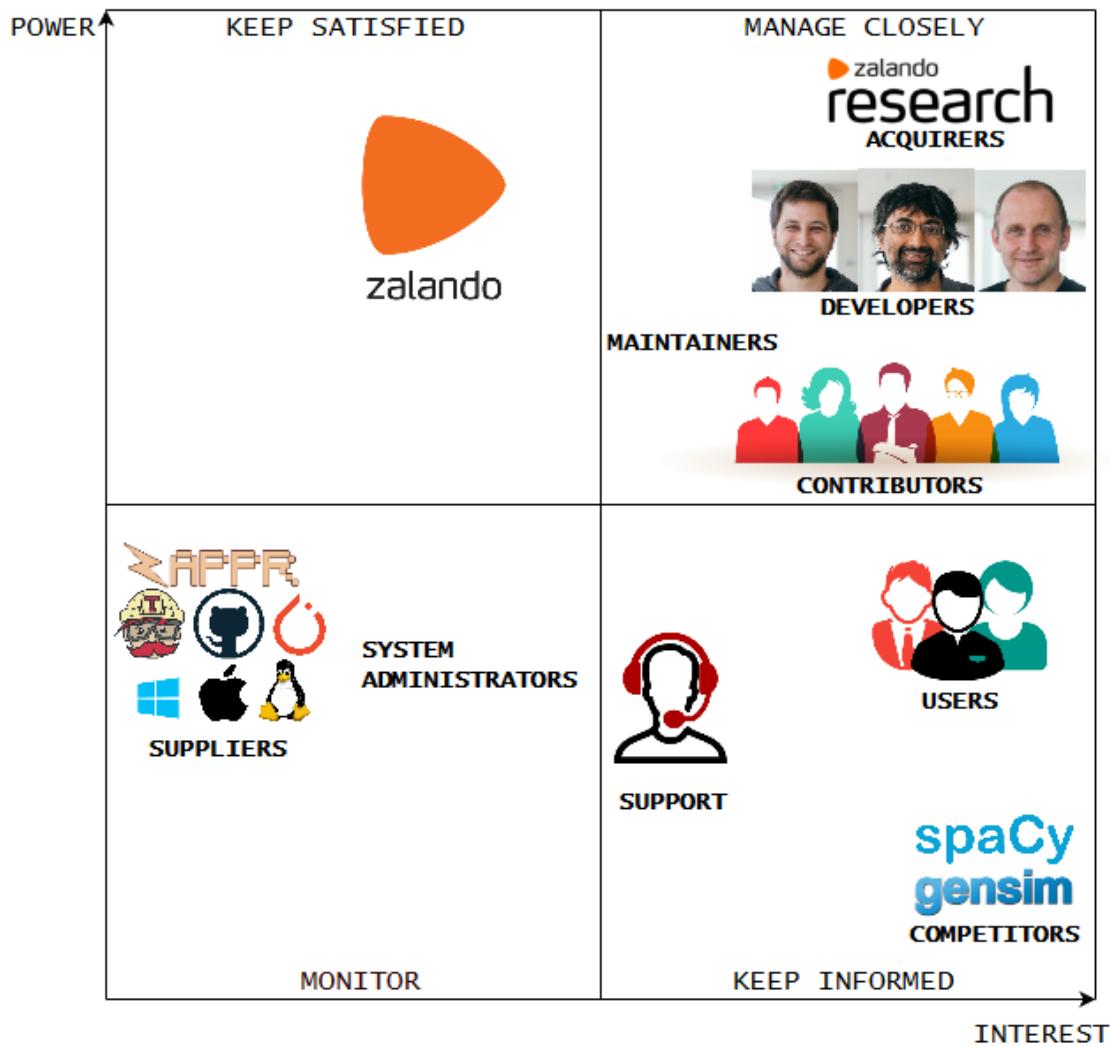
Figure 7.2: Power-Interest grid

The Power Interest Grid for Flair is used to classify the groups of stakeholders necessary to be managed. Most of the stakeholders mentioned earlier are included in the Grid. While users have a high interest in Flair, suppliers do not. Suppliers have more power and should they decide to stop supplying, then Zalando will have to adapt their products accordingly. The contributors, maintainers and developers need to be managed

[2]Mendelow, A. L. (1981, December). Environmental Scanning-The Impact of the Stakeholder Concept. In ICIS (p. 20).

closely because of their inherent interest in the product and the power they hold in influencing design and development.

7.3.4   Relevant people to contact

By analyzing the online community around Flair we found that the following people are the most involved in the project. The relevant people and persons of interests were identified and can be found in Appendix B.

7.4   Context View

Context view describes a software's dependencies, integrations, scope, responsibilities and interface with external entities. In Flair's context, we will show a context view diagram, discuss the system scope and analyze the interfaces with all its external entities.
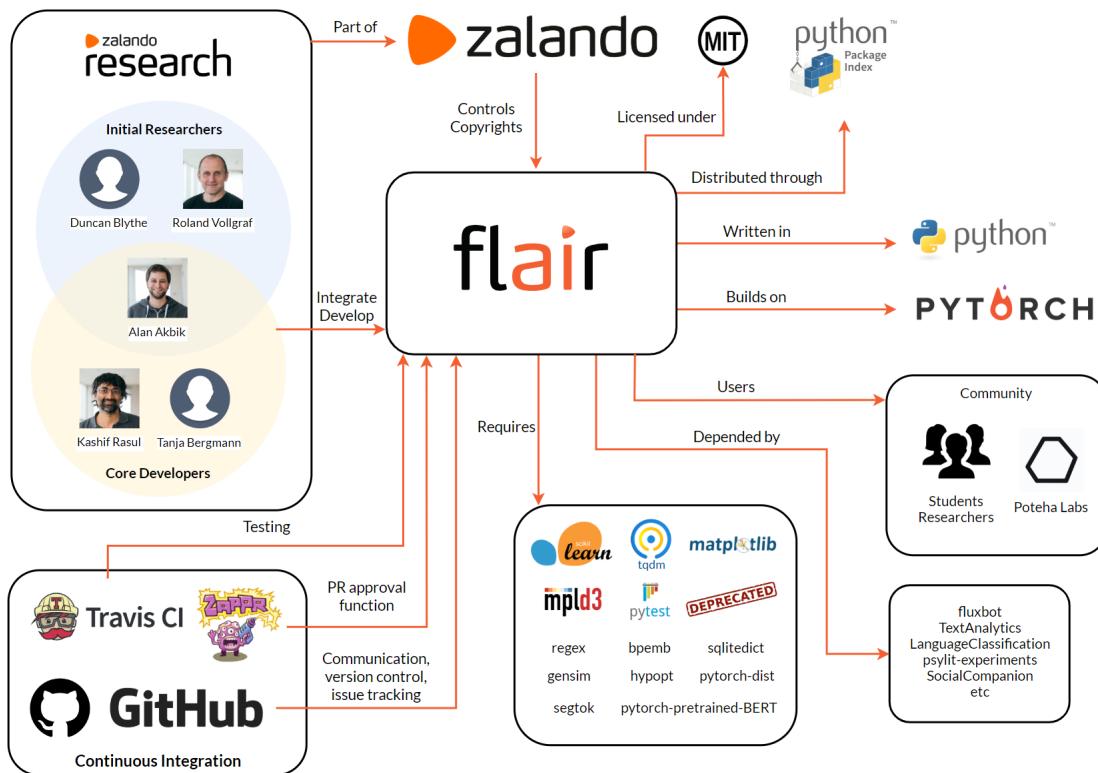
Figure 7.3: Context View diagram

7.4.1 System scope

Flair is a research project of Zalando Research, in the area of NLP. Flair is a cutting edge NLP framework, advancing over all the other existing methodologies. The scope of Flair is well defined and its current version includes the following capabilities [3].

- Built upon Pytorch 0.4+ which is compatible on all operating systems: Windows, MacOS and other Unix-based OSs, and that helps in training your own models.
- Can be easily installed using a pip command.
- It is used to read, understand and learn the contextualized representation of the human language.
- It helps in understanding and classifying the email responses, website comments or customer responses.
- Flair can be trained to recognize fashion concepts such as season, brand, and color.
- Supports state-of-the-art Natural Language Processing models such as name entity recognition (NER), part-of-speech tagging (PoS), sense of disambiguation and classification.
- Supports 36 word embeddings for 31 different languages, character embeddings, Byte Pair embeddings (275 languages supported) and stacked embeddings.
- There were 7 different releases till now and the current version is `Version 0.4.1`.

7.4.2 External entities and interfaces

- Flair is licensed under the following MIT License (MIT) Copyright 2018 Zalando SE.
- It is written in Python 3.6, which provides the framework and programming language.
- It is developed by core committers, developers, community contributors and product managers.
- Users are developers and the organizations that use Flair. Flair is used by Poteha Labs in their research work, by student researchers, and at several in production systems at Zalando.
- Communication channel is used for communcation between developers and users. Flair uses GitHub mainly as its communication channel for interaction with the team. It's also being used for version control. Github is also used for logging issues in the system and for keeping track of them.
- Flair uses Travis CI as a build tool for building its code and thereafter integrating the build. It also uses Travis CI as a testing platform.
- It has interfaces that allows to combine different word and document embeddings, including BERT embeddings, Flair embeddings and ELMo embeddings.
- Flair hosts their bug bounty program privately on HackerOne.

7.5 Development View

Through the theory of development view, we address several aspects of the system development process. Due to the associated complexity of the framework being built, the expertise of the core developers and technologies used are discussed. In the following sections Module Organization, Common Design models and Codeline models are discussed.

The diagram describes a general language model used by Flair. The structure varies based on the kind of embeddings. We start from the bottom, where a sentence is put in as a character sequence into a pre-trained

---

[3]Flair. (2019, February). zalandoresearch/flair: A very simple framework for state-of-the-art Natural Language Processing (NLP).
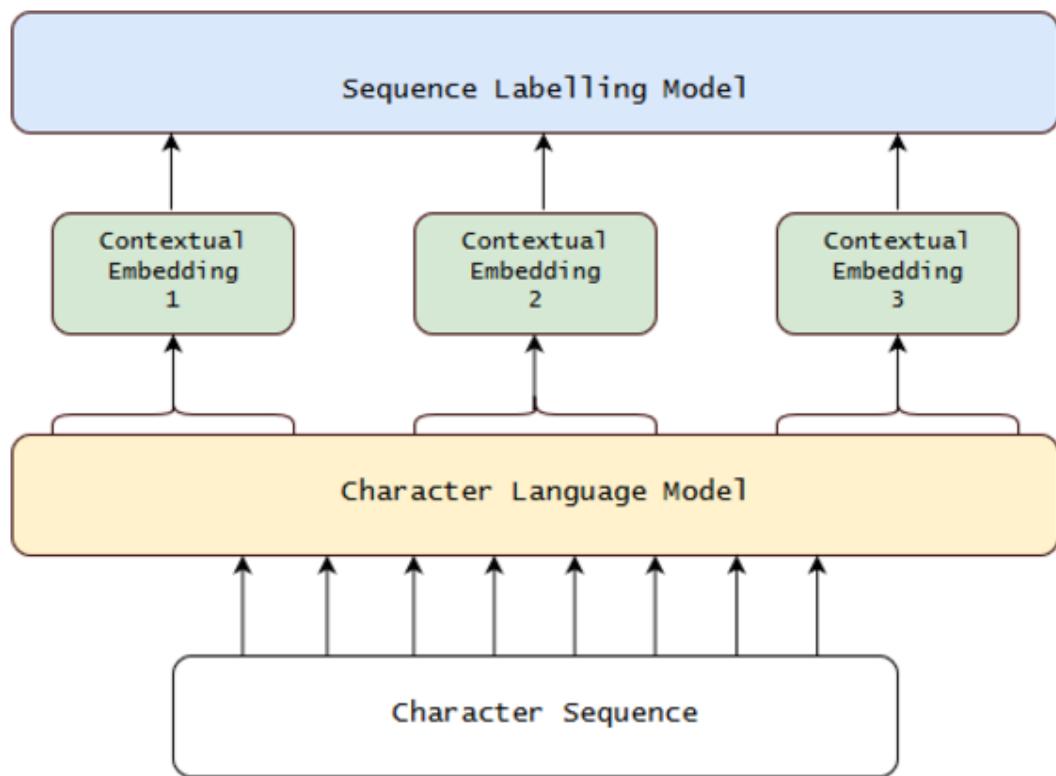
Figure 7.4: Architecture of language model esed in Flair

bidirectional Character Language Model (CLM). From this CLM, they retrieve for each word a contextual embedding by extracting the first and last character cell states. This word embedding is then passed into a sequence labeler, such as a vanilla BiLSTM-CRF, achieving robust state-of-the-art results on downstream tasks.

7.5.1 Module structure and organization

In this section, we detail Flair's module organization, as well as important dependencies between modules, in order to obtain a good understanding of Flair's code structure, based on [4]. The inter dependency of the modules are shown in *Figure 4.2*.

The projects is run directly on a single repository. It is seen that there is high cohesion and low coupling. The high cohesion stems from the `model` and `test` module and the low coupling can be seen from the division of several modules for only certain associated functionalities.

- The `Initialize` package is used to check for compatibility and versions of OS, Pytorch, Cuda and others. The `Data Setup` package is used to setup up data fetchers, model initializers and logging tools.
- The `Preprocessing` module handles text, word and document embeddings-based preprocessing , including tagging, and labeling.
- The `Cache` module deals with utility history for training purposes. It helps encode embeddings into available embeddings list.
- These embeddings can be selected along with the associated `Language Model` in order to determine the classification algorithm.
- `Parameter Optimization` helps select parameters such as document length, batch size, or weights. The results are tested by visualizing plots and training curves.

7.5.2 Common design model

This section covers Flair's **Common Design Model**. Commonality across different versions of Flair is done by defining a set of strict design constraints.

7.5.2.1 Common processing

In Flair there is common processing among different elements in order to simplify the integration of code units and scripts. We discuss the common processing elements here.

- **Initialization** of Flair follows the usage of a particular task such as embedding. Here, NER task is shown. Every embeddings, labeling and language model task has predefined ways of initialization and implementation as given in the tutorials.

  ```
  from flair.data import Sentence
  from flair.models import SequenceTagger
  ```

---

[4]Rozanski, N. & Woods, E. (2011). Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives. Addison-Wesley.
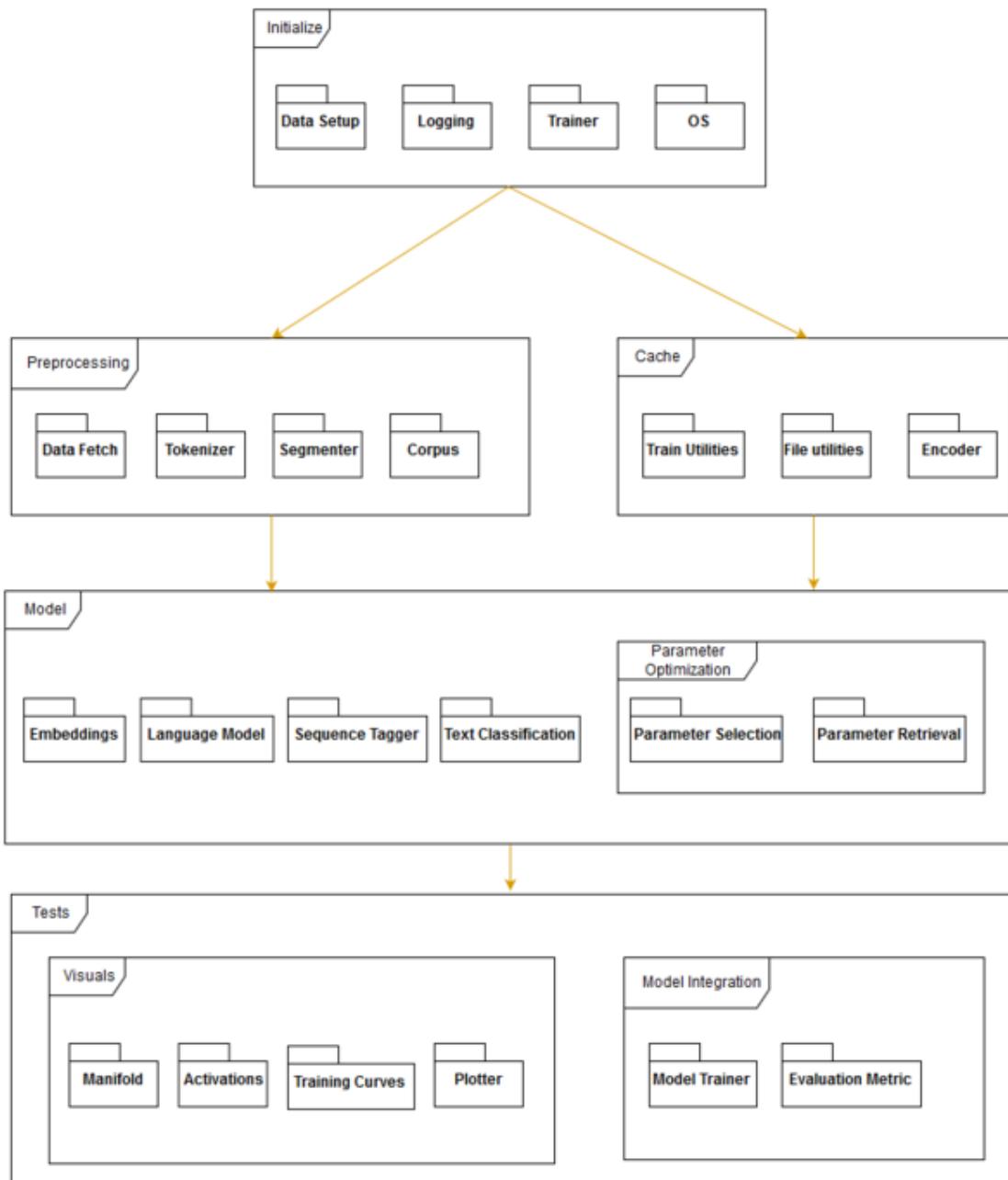
Figure 7.5: Module Organization of Flair

```python
# make a sentence
sentence = Sentence('I love Berlin .')
# load the NER tagger
tagger = SequenceTagger.load('ner')
# run NER over sentence
tagger.predict(sentence)
```

- **Use of third party libraries** exists in the framework of Flair. Flair makes use of third party libraries whenever possible, which we consider to be a standard design approach. We see that in the embeddings, pytorch-pretrained BERT embeddings are used, and certain modules from scikit-learn, gensim and NLTK are also included.

- Through **Instrumentation**, we can measure the performance of a particular metric. The console log and statistics are sent to the developer when tests are run on the integration build using Travis CI.

- Owing to the large processing load that Flair tasks can put the user system in, it is also recommended for the contributors to make use of Google Colab.

- Flair has a clear standardized design methodology and testing procedure when it comes to contributions and changes.

- **Security** issues are acknowledged by Zalando and they try to be responsible by patching quickly. They host a bug bounty program on HackerOne. Vulnerabilities are also posted using this form.

- **Database Interaction** is streamlined by ensuring that newly added corpus are first tested. Once they pass the tests, the datasets are added to the available corpora and the resultant embeddings trained are added.

7.5.2.2 Standardization of design

The Flair repository has the Contributor Covenant Code of Conduct and Contribution guidelines. There is provision of tutorials that provides guidelines for setting up the libraries and embeddings, using pre-trained models, adding your own corpus to train your own models and to train embeddings.

- The tutorials provide details on using libraries and embeddings while creating new embeddings and models. Any steps taken towards issue creation, contributing to issues, making use of pull requests and merging them must be done in concordance with the Code of Conduct and these contributions should fulfill the Contribution and Commit Guidelines.

- Having a global reach allows developers to train and test the language models on different languages other than those already supported by Flair. To maintain a supportive, active community the conduct of conduct states: > In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level of experience, nationality, personal appearance, race, religion, or sexual identity and orientation.

- The coding style employed in Flair requires the developers to follow the Style Guide for Python to maintain conformity. Since the library is still under development, it is necessary to request the developers to set or ask developers to employ any other aspects of coding styles they want to.

- Flair is built using Python 3.6+ as per the requirements. It is mentioned that the expectation from developers is to follow the standards of coding style as defined for each module and library used. The regex guide give tips on regular expressions. The Gensim guide provides tips for using Gensim.

- **Logging** is essential for developers during development to ensure that the system worked as intended and can be understood by another developer.

- **Design Patterns** is used sparingly. The aim of Flair is to focus on performance. As a result, Flair avoids design patterns which in turn acts as an obstacle towards the readability of the code. Different pieces of code working on the same or similar task can look different and would mainly require explanations or comments from the contributors rather than what is mandated by the developers. It makes it more difficult for someone to contribute to the project because the developers might want to see particular code patterns, but these patterns are not communicated to contributors.

7.5.2.3 Standardization of Testing

The standards of testing in Flair are mainly dependent upon integration checks and verification of pull requests that developers make. In this section, the tools used for testing purposes are discussed.

- There are tests specified for a variety of NLP tasks that need to be accounted for. The design pattern followed for testing involves individual task based tests and then integration and build tests. The task based test specifications detail the tests that the partiuclar tasks must pass. Furthermore, the integration to complete the pull requests are tested with Travis CI and Zappr.
- For autoamted testing, Flair uses Travis CI (continuous integration) and it is ran on all pull requests and all commits to a branch to verify that it builds and it passes testing. Regardless of which type of CI run for testing, all CI jobs must pass before the pull request is merged by Flair maintainers.
- The process starts at the *.travis.yml* file in the root of Flair repository. It specifies all the steps for building a test environment. The build lifecycle is made up of two steps, one is install which installs any dependencies required and the other one is script which runs the actual tests. In script, multiple script commands are specified. If one of the build commands returns a non-zero exit code, the Travis CI build runs the subsequent commands as well, and accumulates the build result.
- Using Zappr developers check for patterns in pull requests. At least two approvals from other people including collaborators are necessary conditions. The *.zappr.yaml* file contains other details on how to determine which pull requests are valid based on conditions set.

7.5.2.4 Standard software components

- The dependencies on which Flair depends are standard components required for the funcitoning of the Flair Model. This can be seen in the requirements page. These include gensim, pytorch and other dependent software. All developers aiming to contribute must have these rquired base and dependency software installed in compatible environments.
- Zappr is a GitHub integration built to enhance your project workflow. Built by open-source enthusiasts, it's aimed at helping developers to increase productivity and improve open-source project quality. It does this primarily by removing bottlenecks around pull request approval and helping project owners to halt "rogue" pull requests before they're merged into the master branch.

7.5.3 Codeline models

- **Source code structure**: The complete directory organization of Flair can be seen in *Figure 4.4*.

- **Build approach:** Flair uses *Pull Request Workflow*. The straightforward approach makes it easily understandable for contributors and provide them with naming conventions, files etc. Following a *Pull Request Workflow* helps the developers make contributions via GitHub and allows other contributors to read code changes and make remarks when there are errors or issues in the requested PR. Some PRs have discussion threads, giving more insights about how testing is done and what parameters are checked before a successful merge. The merge is complete only after a final check is carried out in Zappr.

- **Release process:** The release process does not contain any fixed timing, unlike a weekly or monthly release. As of now, no such fixed release process has been mandated by the core developers. However, the last few versions have been released once a month. In fact, Flair now has had a total of 7 releases. The developers are working towards a stable release and therefore are working on the master branch. Bug fixes made and potential speed improvements achieved are highlighted.

- **Configuration management:** Flair is managed directly on GitHub and new versions on Flair are released through GitHub releases. The reliability on GitHub is due to its provision of a competent version control system to maintain the source code. This supports repetition via commits and branches. The configuration structures used are repositories, branches, tags, commits, pull requests, issue trackers and milestones.

- **Contact with the developers**: The initial point of contact starts from Issues. The developer or contributor can discuss and deal with build-specific issues like bugs, errors, crashes and compilation errors. Most of the issues are closed within a week but there are certain issues that take more time.

- **Pull requests**: Pull Requests, allow for contribution and addition of new features, fixes and bug corrections, and optimization of current features. First time contributors are encouraged to look into issues already posted by others in need of help. The official guidelines mention details for contributing with pull requests, checking for issues and explain the Git Commit process. Every pull request is reviewed by Akbik, Schweter and Bergmann.

- **Travis CI** : Flair uses Travis CI to manage continuous integrations to build and test Flair on the fly. When new commits are pushed to the repository, the specifications and commands in the `.travis.yml` file will be run and then project is built. The results of the tests are sent to the developers and maintainers including job logs for specified pull request.

- **Zappr**: Flair makes use of Zappr , a GitHub integraiton developed by Zalando. Zappr makes it easier to manage project workflows. Flair makes use of Zappr to restore and improve code review. Using Zappr, the developers are able to enable/disable pull request approval checks per repository, with the simple flip of a toggle, configure what counts an approval and what doesn't and can provide a configurations file that overrides Github's default settings and endows users with PR approval authorization.

```
flair
├── data_fetcher.py - Datasets, universal dependencies and fetch TaggedCorpus for NLPTask
├── data.py - Dictionary, labels and tokenizer
├── embeddings.py - Abstract base class for all types of embeddings and their combinations
├── file_utils.py - Utilities for working with the local dataset cache
├── hyperparameter
│   ├── init.py - Initializer for parameter optimization
│   ├── parameter.py - Parameters currently used
│   └── param_selection.py - Parameter selection for training
├── init.py - Initializer to start logging and cuda release
├── models
│   ├── init.py - Initializer for language, sequence tagger and text classification model
│   ├── language_model.py - Container module with encoder, a recurrent module and decoder
│   ├── sequence_tagger_model.py - Assignment of label to sequence of embeddings
│   └── text_classification_model.py - Model takes embeddings, puts into LSTM to obtain label
├── nn.py - Abstract base class for all models
├── optim.py - Implements various optimizers such as Adam, Stochastic gradient descent etc
├── trainers
│   ├── init.py - Initializer for Trainers
│   ├── language_model_trainer.py - Language model trainer implementation
│   └── trainer.py - Model trainer implementation
├── training_utils.py - Defines metrics for evaluation and output
└── visual
    ├── activations.py - Highlighter to select activation
    ├── init.py - Initializer for Visualizer
    ├── manifold.py - Splits input in stream of characters, words etc
    └── training_curves.py - Plots training parameters and training weights over time
```

Figure 7.6: Directory organization of Flair and simplified version of the directory strcuture with added explanation

7.6 Technical Debt

Technical debt is a metaphor that describes the immature artifact in the process of software development, that causes an additional rework because of choosing an easy solution over a better approach that would take a longer time to implement. Main reasons for technical debt are leaving the tasks undone, copying and pasting a piece of code, negligence to lessen the code complexity, incomplete unit testing, and lack of proper documentation. In this section, we will discuss the technical debt and its handling by the community. For this purpose, we used two tools, Pylint and SonarQube to analyze the code quality of Flair.

7.6.1 Pylint

Pylint is a bug and quality checker for Python which follows the style recommended by Python style guide PEP 8. Pylint gave the package an overall rating of 7.08 and detected 1168 violations. Although, we note that some of the violations are not major. For example, 418 violations due to lines that are too long, i.e. they have more than 100 characters and 124 violations are caused by invalid variable or argument names that do not conform to the recommended naming style.

As shown in *Figure 5.1*, we see the second most common violation is missing docstrings. It is necessary to put docstrings on functions, methods, modules, and classes in order to preserve maintainability of the software, as docstrings would allow other contributors to better understand what the specific part of the code does.

Pylint also found one important violation, in that the module `embeddings.py` contain 1939 lines, which is almost double the recommended 1000 lines. This also affects maintainability of the code as a very big module is harder to maintain and might end up causing unnecessary dependencies when extended.
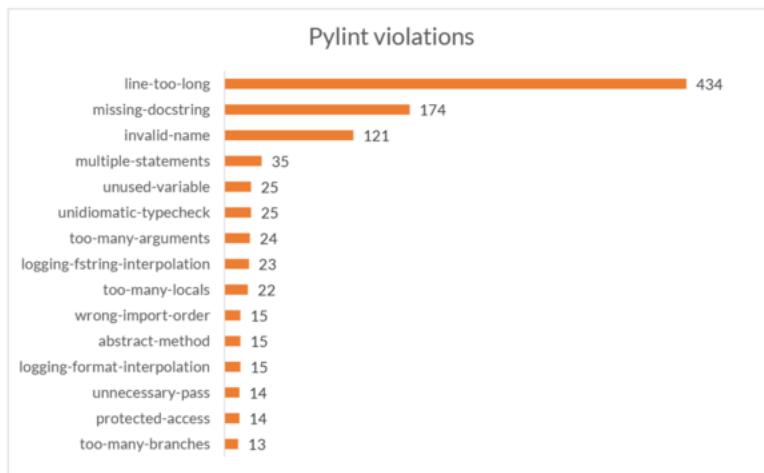
Figure 7.7: Top 15 violations detected by pylint

7.6.2   SonarQube

SonarQube is a framework that detect bugs, code smells, duplicate code, and security vulnerabilities of a program in many languages, including Python.
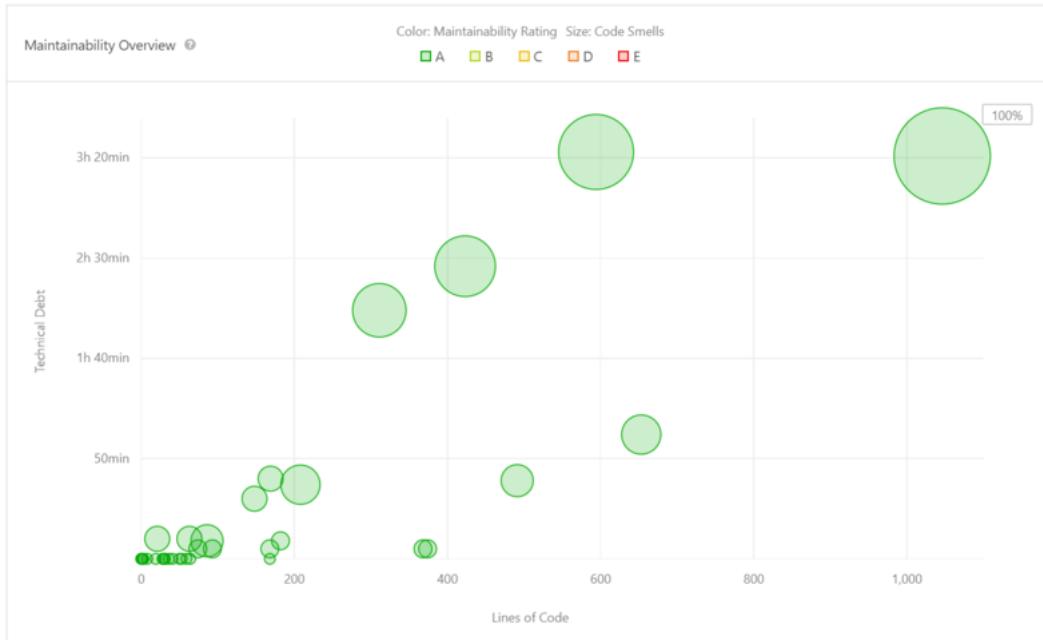
Figure 7.8: Technical debt of each module

7.6.2.1   Bugs

SonarQube detected 18 bugs in Flair codebase. All the 18 bugs are major issues, and hence the code quality has been graded as C. All the bugs are related to the useless assignment to variables. They have no such impact on program functioning but useless variables consume the CPU memory which reduces the performance while training heavy data in Flair. These can be fixed by either removing the redundant variable or by assigning it some different value that was intended for the assignment instead.
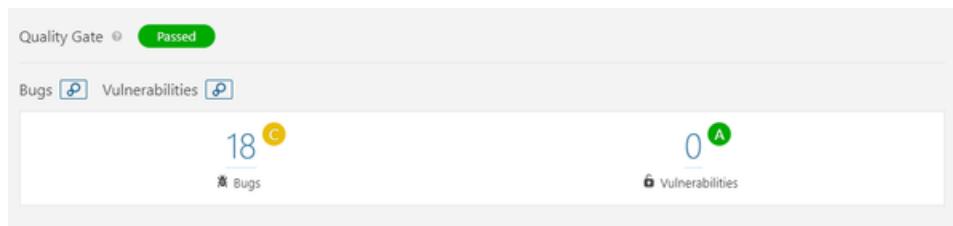
Figure 7.9: Detected bugs

7.6.2.2 Vulnerabilities

Vulnerability detected by SonarQube is 0%, that means code quality is very good from security point of view. The code quality has been graded as A.
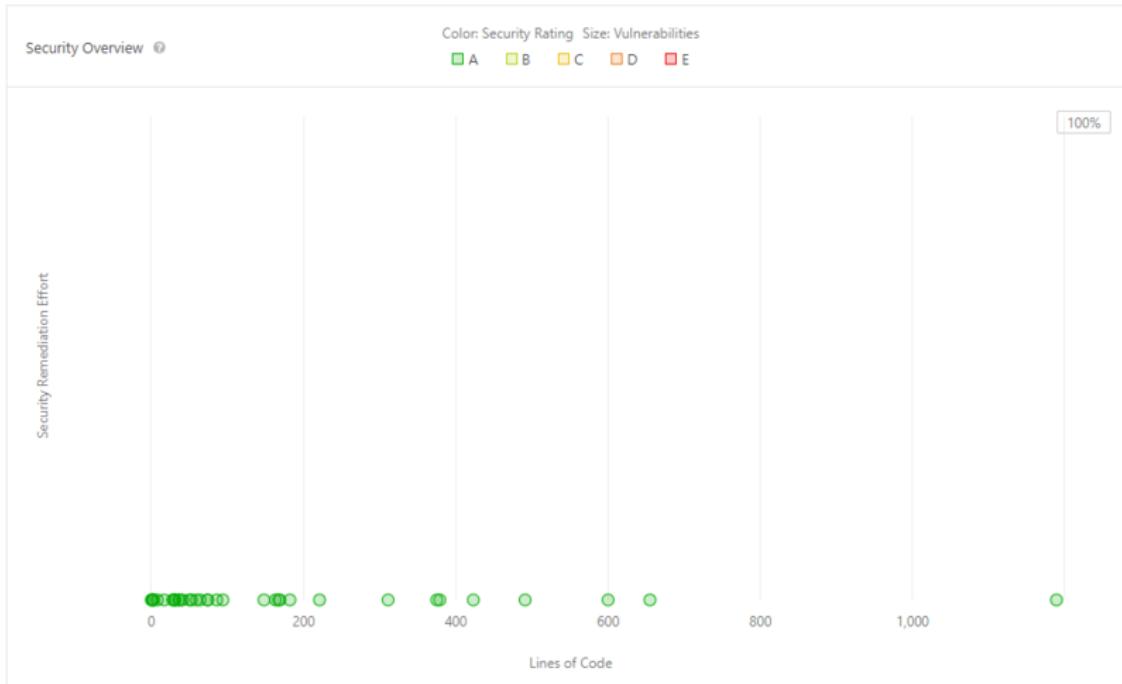
Figure 7.10: Code quality

7.6.2.3 Code smells

Code smells are one of the major contributors to technical debts, and indicates the weakness of the code, and may cause the increased risk of failures or bugs. They are not actually bugs and do not cause any hindrance in program functioning. Total 61 Code Smells detected by SonarQube, most of them are related to Cognitive complexity, that asks for changing the logic of the program because of inclusion of nested loops or conditional statements. As Flair is an NLP library and needs to train large amount of data. So it takes a lot of time to train. These code smells therefore makes the program slower as well as increases the chance of failure or bugs in future. So, fixing these will bolster the performance of Flair.

7.6.2.4 Duplications

One of the worst coding practices is duplication. Duplicating a block of code not only replicates the potential issues but also breaches the coding standards. SonarQube has detected 668 duplicated lines in 5 duplicated files. An overall of 7.2% duplication is detected in Flair codebase. The duplications are due to similar
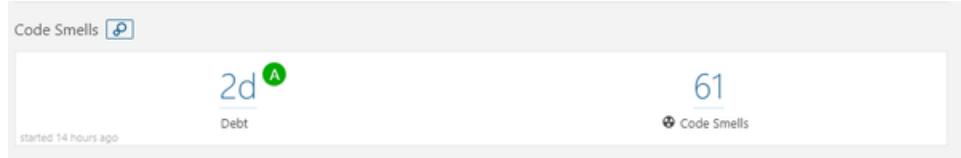
Figure 7.11: Code smells

functions performed for counting the embedding length of different embedding classes. This could be removed by implementing a base function in the abstract base class that could be used by the different child classes.

Figure 7.12: Line duplications

7.6.3   Dependencies on developers

It is noticeable that the product development is highly dependent on two of the core developers, Akbik and Bergmann. A huge difference between the contributions of these two and the other developers of the product can be seen in the graph below. This has created a high risk of project collapse in case one or both core developers leave the team. Ideally, there must not be too much dependency on a single developer, instead majority of the developer community must be equally contributing.

7.6.4   TODO and FIXME comments

TODO or FIXME are the special type of comments that can be added inside the line or block comments for any future references of possible changes, areas of improvements, and optimizations. It is a good indication of technical debt.

We found 2 TODO comments in 2 files in Flair's codebase. This is a very low number for TODO comments. This can be seen in different ways. Firslty, they might be fixing TODO tasks on time and not piling them up in code, which actually is an ideal coding practice. Secondly, they might have removed the comments and have logged an issue instead in GitHub, which actually is a better practice rather than keeping it inside code and creating techincal debt.

The small number of TODO comments is actually an indication of good coding practice, but still these remaining ones needs to be fixed.
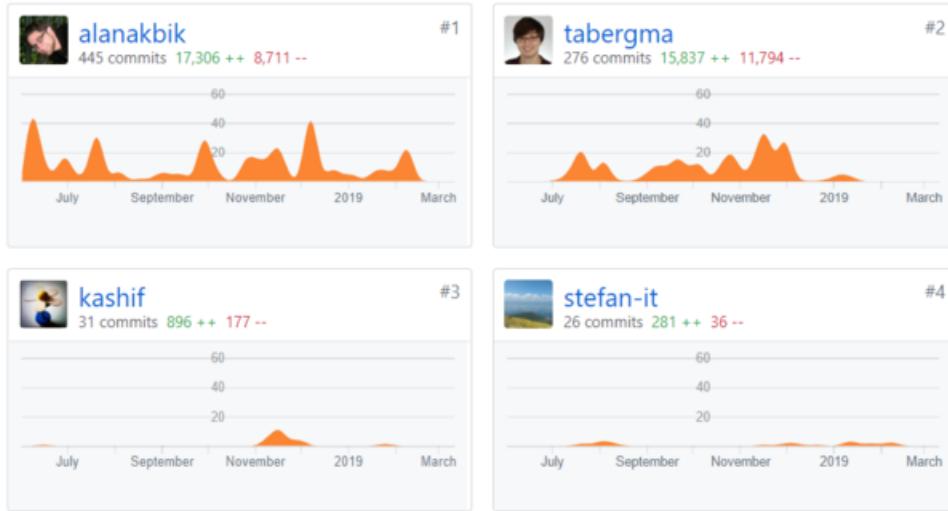
Figure 7.13: Main developers

7.6.5 Testing debt

A part of technical debt is to measure how thoroughly the software is tested. This is also called the **testing debt** of the system. Even though a program may be tested regularly after a change is made in the code, it is important that all modules of the program are covered by the tests. In other words, we need to ensure that the **code coverage** is high.

7.6.5.1 Testing procedure

Flair uses Travis CI to test their product. Each commit to the git repository will trigger a build on Travis CI where a virtual environment on Travis CI's server will be set up. All the required packages will then be installed in this virtual environment, then the test modules under the `tests/` directory will be run in this virtual environment. The code must pass all the tests. There are 12 tests in total.

7.6.5.2 Code coverage

Pytest is a framework that can be used to both test and report the code coverage of a Python program. We used this tool to find Flair's code coverage, then we sent the results to coveralls where the well-tested and less-tested modules are shown in detail.

The result tells us that only 1912 of 4295 relevant lines (44.52%) have been hit at least once by one of the test cases. In *Figure 5.8*, we see that `visual` is the most well-tested module while `trainers` is the least well-tested one.

Although the code coverage seems rather low in most of the files and modules, a closer look into the lines missed revealed that many of them are contained within an `if` block. This means that the `if` conditions are not satisfied during the tests. Considering that Flair is a language processing framework that needs to have

different processes for different languages and the fact that the tests only use resources in English, it makes total sense that the code coverage would be rather low.
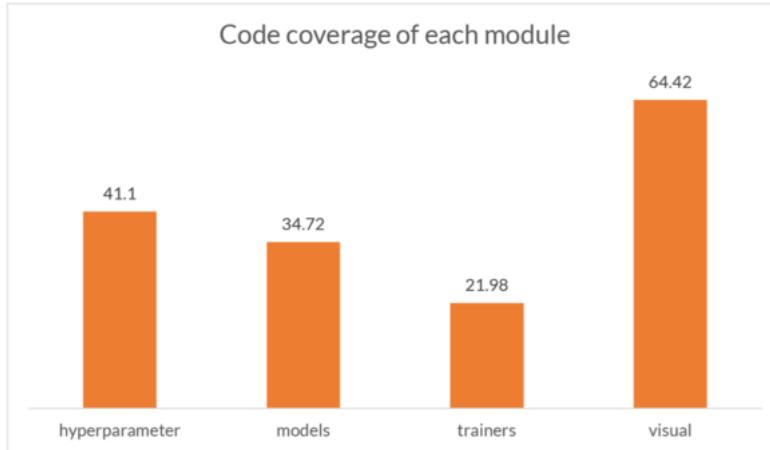
Figure 7.14: Code coverage of each module

7.6.5.3 Actions to improve testing

As discussed earlier that many conditions are not hit due to the separated conditions for each language, the testing could then be improved by augmenting the test resources. Although, we also note that this could drastically improve the testing time.

7.6.6 Discussions about technical debt

There are no discussions about the technical debt between the developers, which is quite shocking. It seems like they are not using any code quality tools. Regardless, the code quality is of the highest level, which actually is a good thing. However, its highly recommended to use one so as to avoid having any future technical debts.

7.6.7 Payment of technical debt

We tried to pay the technical debt by improving the code quality from Grade C to Grade A. This is done by fixing all the 18 major bugs that are detected in SonarQube, to now have 0 bugs in the system. Same source code style is used as in project and its ensured that no new bugs are introduced because of this fix. A pull request (#616) has been created in the master branch of the project Flair, which is still pending and needs to be approved by the owner and then merged. Also, the code build was verified and passed in Travis CI.
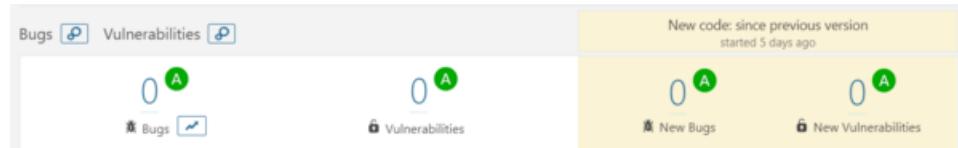
Figure 7.15: Detected bugs after change

Figure 7.16: Pull request

7.6.8 Evolution perspective

Development of Flair was started in the summers of 2018 by Zalando Research and is licensed under MIT [5]. It was made an open source Python project by Zalando SE in June, 2018 [6]. Its based on the research done by Akbik et al. Akbik is the main developer since the start of the project and still is the head developer.

The first draft of the project `Version 0.1.0` was released on Jul 13, 2018 [7]. Since then, 998 commits have been done to the master and 7 different versions have been released, with `Version 0.4.1` being the latest [8]. Technical debt is paid in every release by timely fixing bugs. Also, code is being reorganized if needed, which helps in reducing technical debt like duplications. Documentation is done for every new feature add or for any updation. Different new features were added and improvements were done over the period of time in different releases. Some of the key features of different releases are shown in *Figure 5.12*.

There are 93 issues still open, and 294 are closed [9]. Most of the open issues are of `question` label, but the bug count must be kept zero. Also, there are 6 pending pull requests, which must also be taken into consideration.
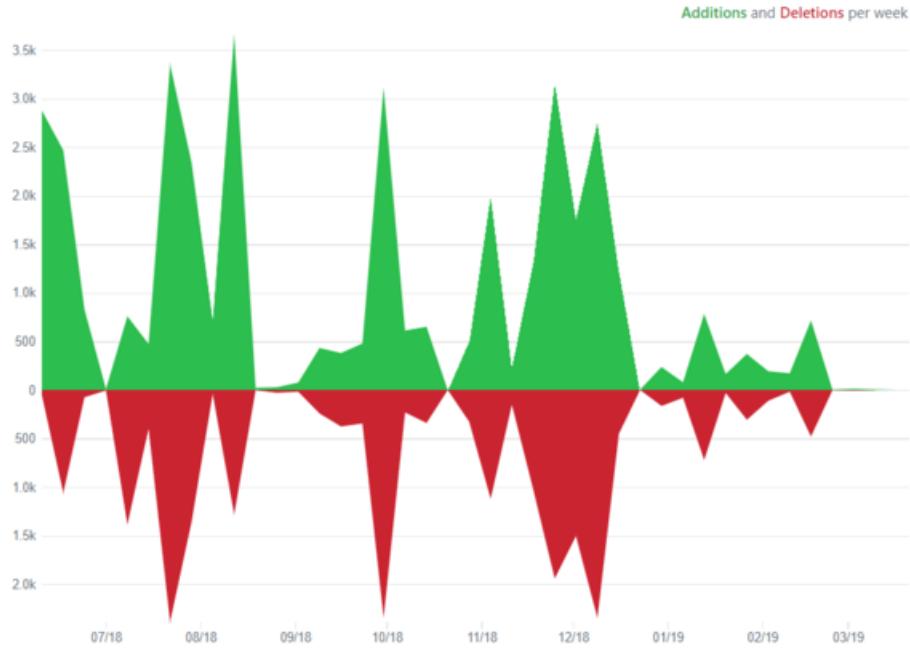
Figure 7.17: Code frequency

---

[5]Flair. (2019, February). zalandoresearch/flair: A very simple framework for state-of-the-art Natural Language Processing (NLP).

[6]Flair. (2019, February). zalandoresearch/flair: A very simple framework for state-of-the-art Natural Language Processing (NLP).

[7]Flair. (2019, February). zalandoresearch/flair: A very simple framework for state-of-the-art Natural Language Processing (NLP).

[8]Flair. (2019, February). zalandoresearch/flair: A very simple framework for state-of-the-art Natural Language Processing (NLP).

[9]Flair. (2019, February). zalandoresearch/flair: A very simple framework for state-of-the-art Natural Language Processing (NLP).

Figure 7.18: Flair's releases

7.7 Functional View

Functional view mainly describes systems architectural elements. It explains about primary interactions between the functional elements, their exposed interfaces and their responsibilities [10].

7.7.1 Functional capabilities

Functional capabilities define what the system is required to do and what it is not required to do [11]. Flair has many capabilities and some of the most vital ones are listed in the table below.

Modules	Description
embeddings.BytePairEmbeddings	Supports 275 languages. Very useful for training small models. This can also be used in sequence tagging.

---

[10]Rozanski, N. & Woods, E. (2011). Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives. Addison-Wesley.

[11]Rozanski, N. & Woods, E. (2011). Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives. Addison-Wesley.

Modules	Description
embeddings.TransformerXLEmbeddings	As proposed by Dai et al. [?], this embeddings address the limitation of fixed-length contexts, as a notion of recurrence is introduced by reusing the representations from the history.
embeddings.ELMoTransformerEmbeddings	Uses a bidirectional recurrent neural network to predict the next word in a text.
embeddings.DocumentRNNEmbeddings	This class allows to choose which type of RNN to use.
embeddings.FlairEmbeddings	Trained without any explicit notion of words and model words as sequences of characters. Same word will have different embeddings depending on its contextual use.
embeddings.WordEmbeddings	Supports pre-trained FastText Embeddings for 30 languages: English, German, Dutch, Italian, French, Spanish, Swedish, Danish, Norwegian, Czech, Polish, Finnish, Bulgarian, Portuguese, Slovenian, Slovakian, Romanian, Serbian, Croatian, Catalan, Russian, Hindi, Arabic, Chinese, Japanese, Korean, Hebrew, Turkish, Persian, Indonesian.
data_fetcher.NLPTaskDataFetcher	This can be used to load different datasets using the `load_corpus()` function.
trainers.ModelTrainer	This is used to train a user's own language model embeddings. It supports multi-dataset training using a `MultiCorpus` object.
models.LanguageModel	Container module that has an encoder, a recurrent module, and a decoder.

Modules	Description
models.TextClassifier	The module takes word embeddings, puts them into an RNN to obtain a text representation, and puts the text representation in the end into a linear layer to get the actual class label.
models.SequenceTagger	Contains several pre-trained English and Multilingual models, which include Named Entity Recognition (NER) and Part-of-Speech Tagging (POS). The function `load()` can be called with the appropriate string to load a model.

Table 6.1: Some of Flair's most vital modules

7.7.2 Functional structure model

Flair has three major functional elements: embeddings, model trainer, and train and file utitilies. Flair uses its train and file utilities to read corpus objects with which a model can be trained. There are several different embedding classes that Flair provides, but all of these embeddings can be classified into three types: BERT, ELMo, and Flair embedding classes. The user can choose to perform a named entity recognition, part-of-speech tagging, and sequence tagging using one of these embeddings.

7.8 Performance and scalability perspective

7.8.1 Performance

Task	Language	Dataset	Flair	Previous best
Named Entity Recognition	English	Conll-03	**93.18**	*92.22 (Peters et al., 2018)*
Named Entity Recognition	English	Ontonotes	**89.3**	*86.28 (Chiu et al., 2016)*
Emerging Entity Detection	English	WNUT-17	**49.49**	*45.55 (Aguilar et al., 2018)*
Part-of-Speech tagging	English	WSJ	**97.85**	*97.64 (Choi, 2016)*
Chunking	English	Conll-2000	**96.72**	*96.36 (Peters et al., 2017)*

Task	Language	Dataset	Flair	Previous best
Named Entity Recognition	German	Conll-03	**88.27**	*78.76 (Lample et al., 2016)*
Named Entity Recognition	German	Germeval	**84.65**	*79.08 (Hänig et al 2014)*
Named Entity Recognition	Dutch	Conll-03	**90.44**	*81.74 (Lample et al., 2016)*
Named Entity Recognition	Polish	PolEval-2018	**86.6** (Borchmann et al., 2018)	*85.1 (PolDeepNer)*

Table 7.1: Comparison with state-of-the art

As shown in *Table 7.1*, Flair outperforms the previous best in different range of NLP tasks like Chunking, Named Entity Recognition, Part-of-Speech tagging. It supports best embedding configurations for each task. The performance is measured over evaluation datasets, and the F1 score for each dataset is calculated by averaging over five runs [12]. In the latest release `Version 0.4.1`, there is a huge improvement in the training speed compared to the previous versions of Flair (2x training speed for language models)[13].

Flair achieved optimal results in text classification outperforming `Facebook's FastText` in accuracy & `Google's AutoML Natural Language` in training speed on `Kaggle's SMS Spam Detection Dataset` [14]. Flair model achieved an f1-score of 0.973 after 10 epochs while `FastText` achieved f1-score of 0.883 [15]. It beats `AutoML` in terms of training speed and achieved slightly better accuracy than it [16].

7.8.2 Scalability

Scalabiity with regards to Flair involves handling larger corpus of text in a quick manner with appropriately scaled usage of hardware. It is difficult to gauge the growth of Flair considering the increase in data set size. Although, until now it has been able to handle vast sizes of corpora. The model has had many scalability improvements over throughout its releases.

Scalability Issue	Details
Language model optimizations	Lowers default dropout rate and only send mini-batches to GPU for better GPU memory usage.
Data loading	Implemented Pytorch capability to load in parallel Dataset and DataLoader classes which cuts training time in half.
Hardware requirements	Usage of a centralizeed hardware resource is proposed similar to Google Colab.

---

[12]https://gitlab.ewi.tudelft.nl/in4315/2018-2019/TI3125TU-swa-12-flair/swa-12-flair/blob/master/report.md#ref3
[13]https://gitlab.ewi.tudelft.nl/in4315/2018-2019/TI3125TU-swa-12-flair/swa-12-flair/blob/master/report.md#ref3
[14]Magajna, T. (2018). Text Classification with State of the Art NLP Library — Flair.
[15]Magajna, T. (2018). Text Classification with State of the Art NLP Library — Flair.
[16]Magajna, T. (2019). How to Beat Google's AutoML - Hyperparameter Optimisation with Flair.

Scalability Issue	Details
Implementation changes	Faster inferencing by presorting sequences and embeddings are cleared after processing each batch, thereby saving a lot of RAM.
Optimizations	Changes made to the hyper parameters and other memory tweaks.
External dependencies	Since the framework depends on CUDA for training, changes made to their implementation must be reflected in Flair. This included memory and parameter tweaks.

Table 7.2: Scalability changes

From a technical standpoint, the data must no longer be loaded in big batches onto RAM directly, instead, it must be asynchronously loaded to the disk. Other tasks such as regression must also be supported by the framework and addition of such features is made easy. Multiple-tasks learning can be included along with multi-GPU support with the changes to the new CUDA semantics. These features while make scaling of Flair easier.

7.9   Conclusion

Flair provides an innovative framework for natural language processing. Within a short period of time, Flair has grown fast and challenges existing NLP frameworks. This growth has been possible through their specific code of conduct for code writing and analysis, constant code quality, and well structured documentation. We have analyzed Flair through various perspectives and viewpoints to have an insight on its internal workings. Its well-built architecture, integration between different modules though maintained common processing, standardization of design and testing has helped Flair to provide a good performance.

7.10   Appendix

7.10.1   A. Pull requests analysis

The pull requests made through github are analysed using standard code of conduct. The code of conduct includes naming conventions, way of codes, standard of code and versions of sources used. This strict follow up of the conduct makes the original code and merged codes quite readable and easily understandable also. Whenever new modules are added, emphasis is given in order to update the documentation as well. As mentioned earlier, integrators check for code standard and versions of PR before the merge. On the other hand, the opened issues that are solved by other contributors in separate PR are closed. This does lead to less confusion about PRs. The closed PRs are sometimes those who are considered to be not at all matching their base mark and also totally unstructured. Also, naming conventions sometimes become a major reason for closed PRs.
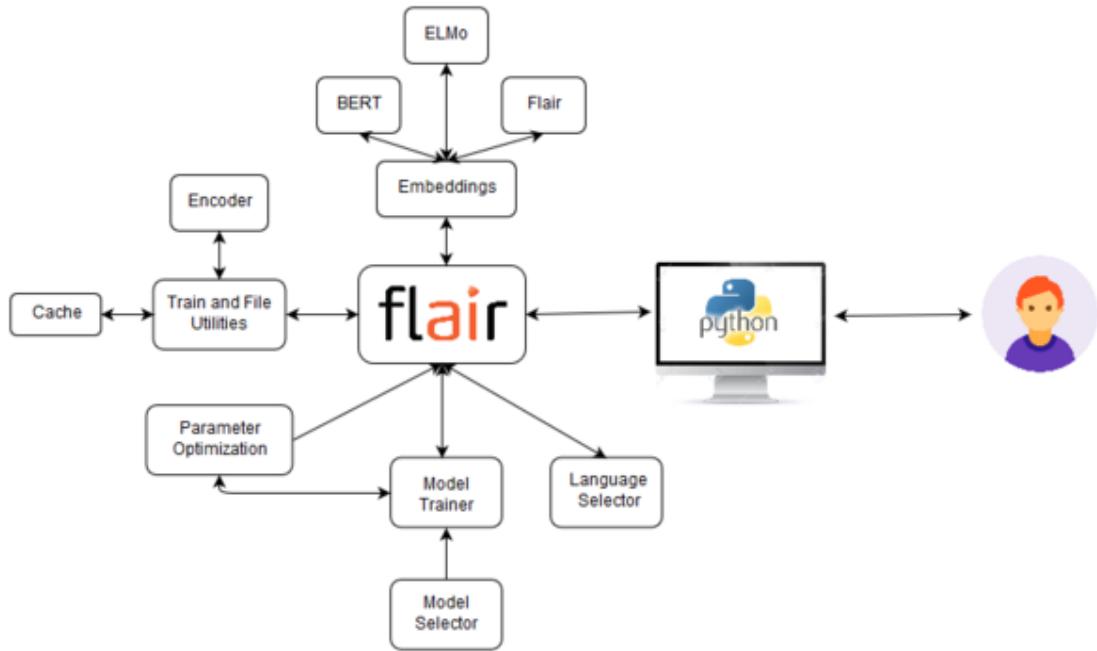
Figure 7.19: Flair's functional structure model diagram

ID	Pull Request	Status	Details
#399	Add new ELMoTransformerEmbeddings class	merged	In this pull request, the contributor adds new class named 'ELMo-TransformerEmbeddings' for Parts of speech tagging. In NLP, PoS tagging is an important step towards processing. The new class gets embeddings from a transformer-based ELMo model from another contributor @brendan-ai2 which one uses for training. The allennlp library is used for PoS tagging. Though the version of allennlp used was not yet updated in the flair main system, so testing was separately done and this successful testing outside the main system with latest version of allennlp suggested a good merge. So, it is merged but as experimental.

ID	Pull Request	Status	Details
#523	GH-522: Transformer-XL embeddings	merged	This PR adds a new embedding class for *Transformer-XL* model. Initially the tokens from a sentence were returned with tokens tag only bur this also included enumerated token tags. Also, the F-score of the tests increases a lot with the new embedding and the PR was accepted.
#473	GH-438: added byte pair embeddings	merged	It embeds BPEmb library for for sequential tagging. It adds the *BPEmbSerializable* (BPEmb) class which embeds sequential tagging using torch library in tensors. It was accepted because the sequential tagging can also be deserialized when required like updating the cache_dir variable. The changes are merged after the development of the software and launch.
#164	Added class-based metrics	merged	This basically improves the metric class through updating the `TextClassifierTrainer` function and removing `calculate_class_metrics` and the `calculate_micro_avg_metric`. The changes are merged after the development of the software and launch.

ID	Pull Request	Status	Details
#538	Adds Attentional BiLSTM in the Text Classification architecture	merged	Flair used standard BiLSTM model for text classification whereas this PR introduces Attentional BiLSTM model for this. The basic introduction faces error especially for training. So, the new PR generated with code updates were accepted with Attentional BiLSTM model. The PR was merged after launch of initial version.
#557	Fix permission error in Windows	merged	This fixes issue #548. It was unable to delete the file which was previously opened (thus, marked locked). More precisely, `mkstemp()` opens the file, which then creates a lock on the file, therefore prohibiting Windows from deleting it.The updated function mktemp() does not do this. SInce it does not create the lock, files can be deleted when necessary.
#551	GH-550: Fix bert model check	merged	This PR fixes an #550 which occurred due to a code fix mistake from the #523. The issue raised becaused the code did not include bert_model in BERT_PRETRAINED_MODEL_ARCHIVE_MAP.The inclusion of the model was tested successfully and merged.

ID	Pull Request	Status	Details
#555	Remove BERT_PRETRAINED_MODEL_ARCHIVE_MAP check to support Custom models	merged	This issue was due to version change of pytorch-pretrained-bert. This issue was also raised at #554. the BERT_PRETRAINED_MODEL_ARCHIVE_MAP was removed from the model as solution.
#71	GH-38: Add label class for sentences to hold label name and confidence value	merged	It introduces a new class `lebel` which creates new label for the sentence. Each label has name and confidence value (between 0 and 1). It is used to predict return label/tag plus confidence value. It was merged after the development and launch of the system.
#519	Added PubMed embeddings computed by @jessepeng	merged	A character LM over PubMed abstracts was calculated by @jessepeng. This PR embeds the `PubMed` to the system with `FlairEmbeddings` model. Additionally, it needed a few details like batch size, layers, hidden size, train size etc for the documentation which was added by this PR. It was merged after the development and launch of the system.

ID	Pull Request	Status	Details
#88	embeddings.py 213 214 a warning	closed	GitHub user @wuwingzhou828 proposed a change to certain lines of code in the section where the standard static word embeddings are defined including GloVe or FastText. It pertained to an if statement, where in case a certain precomputed word embedding already existed that particular embedding is fetched using the `token.test` parameter. Initially the change is accepted by Alan Akbik but merging is held on until all test were completed. However, the user made some errors in the correction as well and therefore the request was closed by Tanja Bergmann.

ID	Pull Request	Status	Details
#490	GH-68 Bert update	closed	Pull request was made to clear issue #68 regarding the implementation of a pipeline by a transformer based model implemented by OpenAI. The issues pertaining to addition of transformer models are discussed. They added Bert embeddings to the library. As a result the pull request refers to changes in the files setup.py and pytorch, thereby updating the BERT version. However, a newer version updated by user stefan-it was released with the updates made in Flair version 0.6.1.
#22	GH-226: add safeguard for non-tokenized input	closed	User @jfilter opened a request to address the issue of a `ValueError` if the input is not whitespace tokenized and no tokenizer is used and therefore causes an infinite loop. However, the issue was already fixed in GH-232: multi dataset training #236

ID	Pull Request	Status	Details
#494	calculate macro-f-score as average of class f scores	closed	There was probably an error in calculation of macro F1. Flair is taking average of class precisions and recalls and then calculates F1. As far as is known, it should be F1 calculated over classes and then averaged. This issue is fixed in #521. Hence the pull request is closed by Alan Akbik. This pertains to the initial fix not working in updated version which was then fixed in a later pull request. Therefore this pull request is deprecated by the fixes made in the newer pull request.
#246	GH-243: dataset downloader	closed	This refers to the dataset downloader and updates made to the dependencies and corpora. The DataFetcher would now download corpora by checking its availability. However, the updated library did not pass one of two tests to check for validity and the request was closed.

ID	Pull Request	Status	Details
#13	GH-12: naming conventions	closed	Request made by Alan Akbik to update the naming conventions associated with module imports, training classes and embedding classes. Changes were deprecated with new pull request that was merged - GH-12 naming conventions #14. This allowed for better class and file matchings and easier imports of embeddings.
#455	master	closed	Pull request to make several commits made in personal branch to the Flair master branch. It was a mistake on the part of the user gccome and requests were made to the maintainers to delete the particular PR. Good example of the role of maintainers and support staff.

ID	Pull Request	Status	Details
#443	GH-387: long text embeddings	closed	References to an issue reported by several users that when training on GPU with very long sequence of words, there is an out-of-memory error. This is due to the batch being padded with the length of the longest sequence which is then put into the LSTM. Solved this by breaking the long string in chunks that are fed sequentially into the Language Model. This therefore results in the removal of the long text embeddings branch due to deprecation and failure to comply with tests.
#131	GH-61: Visualizations	closed	Initially opened to address visualization requests such as visualization of weight traces during training and states in character LM. Pull request deprecated by changes made in GH-61: Visualizations #132. Being able to visualize the training patterns allows the users to better understand the process of training the embeddings.

ID	Pull Request	Status	Details
#21	GH-19: simplify sequence tagger	closed	Initially made to add learning rate scheduler to SequenceTaggerTrainer for easing the use of Sequence Tagging. Also provided auto-spawn into GPU for the models removing the move-to-GPU commands that would otherwise be required that are quite cumbersome and redundant.

Table A.1: 10 merged and 10 closed pull requests in Flair

7.10.2   B. Contact persons

Name	GitHub	Role	Twitter
Alan Akbik	@alanakbik	Zalando, Developer and Maintainer	@alan_akbik
Duncan Blythe	@blythed	Zalando, Developer	-
Henning Jacobs	@hjacobs	Zalando, Head of Developer Productivity	@try_except_
Tanja Bergmann	@tabergma	Zalando, Developer and Maintainer	-
Per Ploug	@perploug	Zalando, Open Source Manager	@pploug
Kashiv Rasul	@kashif	Zalando, Developer	@krasul
Stefan Schweter	@stefan-it	Munich Digitization Centre at the Bavarian State Library, Developer	-

Table B.1: People most involved in Flair's development

7.11   References

Chapter 8

Flutter

Yanni Chiodi, Mathieu Post, Emiel Rietdijk, and Eva Verboom

8.1   Table of contents

161

8.2   Introduction

Flutter is a Google initiative to help developers build native apps on iOS and Android from a single codebase fast and simple. Recently they also released Fuchsia and added it as a compatible operating system. While the project initially set out to develop a UI framework, they switched to constructing an application SDK as users desired a complete implementation. The SDK contains mainly widgets that can be used to build the application UI. A widget is a rigid description of the user interface. All graphics including animations, shapes and text are created with the use of widgets and more complex ones can be created by combining simpler widgets.

The basis of the open source community that is working on Flutter consists of a team of engineers working for Google, though there are some non-Google contributors that also have added a fair share of codelines to the project. Flutter is a very active and fast growing project, maintained and guided by the core team. In December 2018, the first stable version of Flutter, v1.0, was officially released. Afterwards, the team grew tremendously and they keep growing to be able to maintain and update the toolkit.

This chapter aims to describe the architecture of Flutter based on different views, characterize the stakeholders and provide details of the evolution Flutter went through. In doing so, this chapter will not only cover the technical aspects of the architecture, but also provide a broader view of the project. Then we have a section on technical debt, which shows where improvement is still required and discusses how this is handled by the developers of Flutter. The final section will discuss the aspects of Flutter regarding operations and implementation. Apart from doing research based on the GitHub repository [4], Flutter documentation [2], the wiki [12] and other online sources, we also contacted the Tech Lead of Flutter, Ian Hickson (@hixie on GitHub), to answer some of our questions.

We have also contributed to the project by fixing an open issue. At the time of writing this is not yet merged or closed.

8.3   Stakeholders

The stakeholders of a project can be defined as the people, organizations, groups and/or companies that have interest in the realization of the project. Stakeholders can affect (or be affected by) the actions, objectives and policies concerning the project.

8.3.1   Rozanski and Woods classification

Rozanski and Woods describe eleven stakeholder classes in their book "Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives" [1]. In this section the types of stakeholders that apply to Flutter are discussed.

8.3.1.1 Acquirers

The acquirer of Flutter is **Google Inc.** They employ the core team working on Flutter which built the base of the framework and maintains it.

8.3.1.2 Assessors

There are no obvious assessors of this project. Flutter is only licensed under the BSD License [5]. Therefore, Flutter can be changed and shared among others as long as the contributors are given appropriate credit and all changes are indicated properly. Though Flutter works with a lot of software components that have their own licenses [6]. As every pull request is checked by at least two members of the core team, they are also responsible for checking the conformance of the code to standards and legal regulation.

8.3.1.3 Communicators

The system is explained to stakeholders through the corresponding website [7] by members of the core team. On the website you can find videos, showcases, tutorials and basic documentation explaining the development and the possibilities of contributing to the application. Furthermore, contributors and committers are communicators by default, as documentation and justification is a requirement for every contribution. Contributors communicate with each other over numerous channels, like Gitter, Slack or Google Groups). Google also organizes events where developers working on the Flutter project can meet.

8.3.1.4 Developers/Maintainers

There are a total of 34 contributors to the Flutter project, which are listed in the AUTHORS file on GitHub. Here Google Inc. is listed as one of the contributors. This is the core team that is working for Google and is involved in the Flutter project. They also do the maintenance, together with the other trusted contributors. Some of the most frequent contributors are non-Google engineers.

8.3.1.5 Suppliers

Flutter is written in Dart [3] for iOS, Android and Fuchsia operating systems and can be used by anyone to build their own mobile applications for free. GitHub hosts the repository containing the code of the project.

8.3.1.6 Support Staff

Most support is handled by Google itself, though contributors also help each other out. Apart from the clear examples, documentation and tutorials on the website, Google also encourages people to join the discussion on one of their communication channels and everyone is free to ask questions on Stack Overflow or send them an email at flutter-dev@googlegroups.com if there are any additional question.

8.3.1.7 Testers

Flutter uses both native Dart tests and special Flutter tests which are updated with every bug fix and run on every commit. Afterwards, a test coverage report of the project is created and can be reviewed at coveralls. These tests are mostly written together with new or changed code by the developer that is contributing the change.

8.3.1.8 Users

The users of Flutter are the commercial and private developers that use the system to create their own mobile applications. They are also invited to contribute to the project in order to realize their needs within the project.

8.3.2 Other Stakeholders

Besides the classes of stakeholders defined by Rozanski and Woods, there are four other classes that can be identified within the Flutter project.

8.3.2.1 Media

There are several websites that posted about the Flutter project like One More Thing [8], The Verge [9] and Maxdoro [10]. The website It's All Widgets [11] even allows developers to submit their Flutter build applications to showcase them to others as examples and inspiration.

8.3.2.2 Competitors

Currently there are many different systems that can be used to develop mobile application, such as Apache Cordova, Adobe PhoneGap, Iconic and Facebook's React Native.

8.3.2.3 End Users

The end users of Flutter are the customers that will eventually download and use the apps of the companies that use Flutter to build their own mobile applications.

8.3.2.4 Partners

Some of the companies that are using Flutter, reach out to the core team for a partnership. For example when they believe that they need additional features and they are willing to invest in that, they can build that together with the core team and other members of the community. When there is a close relation between Flutter and the customer, they try to give a higher priority to issues related to such a partnership.

8.4 Power versus Interest

A graphical representation of the involvement of all stakeholders is shown in the figure below. The core team has the most power and interest in Flutter. Though, according to Ian (Tech Lead) the other contributors are as much a part of Flutter, thus they are close to each other in the grid. Google does have a lot of power, but as Flutter is not one of their main products, their interest is lower and they are not directly one of the key stakeholder. Furthermore, there are some developers that use Flutter for application development, but do not contribute. They have high interest and should be kept informed. Their power lies in giving feedback on usability and proposing new features. Lastly, Dart and the operating systems the SDK is built for are minimal effort stakeholders.
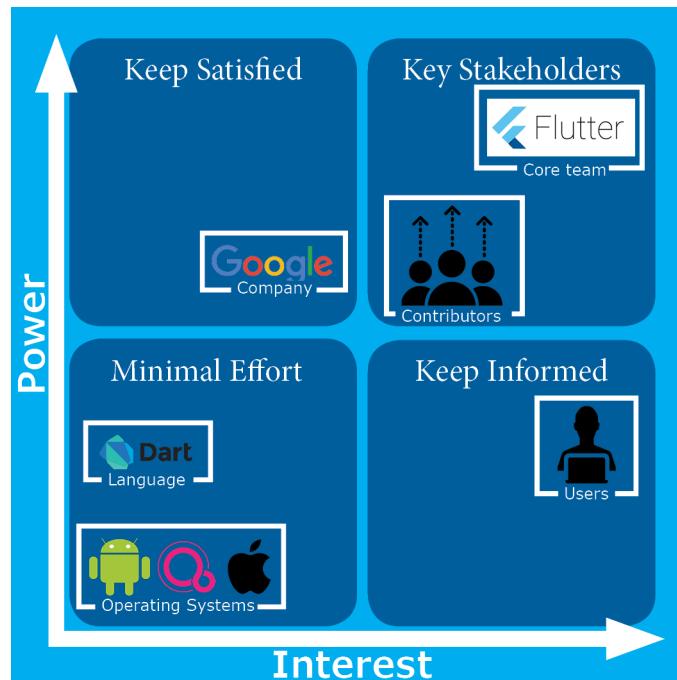
Figure 8.1: Power vs. Interest Grid of the Flutter Project

8.4.1 Integrators

The integrators are responsible for project quality and longevity. They communicate requirements to contributors, perform code reviews, lead discussions for new features and trade-offs and solicit new contributors. Ultimately, it is the integrator's role to decide whether contributions are accepted or rejected.

The integrators for the Flutter project are the core team members, mostly developers that work at Google. Their decision to merge is based primarily on the quality of the code, but they also tend to focus on whether the contributor follows the Flutter style. A very common challenge for them is that the quality of the contributor's code/style/documentation is not up to par with the Flutter standards. As a consequence, the

integrators have to spend a lot of effort reviewing and giving feedback. Even 'trivial' things like spelling mistakes in documentation have to be corrected.

8.4.2   Decision Making

To gain more insight on the way decisions are made within Flutter, the most discussed merge requests of the project were analyzed. When handling new contributions to the platform, there are a few key things that are discussed before accepting/rejecting issues:

- The conformation of the code and documentation formatting with the Style guide
- Questions raised by the developer of the issue and by reviewers on the implementation
- Trade-offs and/or compromises about how to implement the fix
- Readability and as few code and as few dependencies as possible are recurring issues in discussions.

8.5   Context View

This section describes the context view of the Flutter project. The context view includes the relationships, dependencies and interactions of Flutter with the environment.
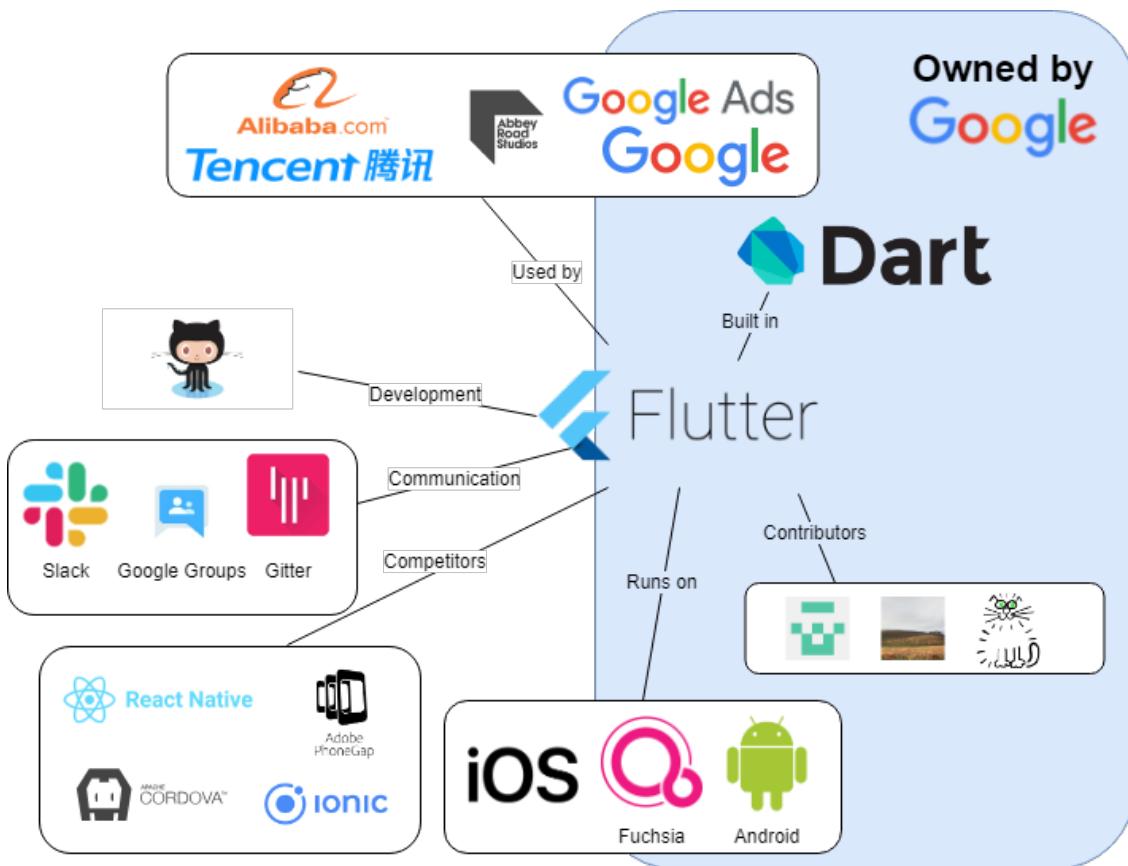
8.5.1   System Scope

The description of Flutter on their repository on GitHub is as follows: *"Flutter is Google's mobile app SDK for crafting high-quality native interfaces on iOS and Android in record time. Flutter works with existing code, is used by developers and organizations around the world, and is free and open source."*

Flutters main objective is to help other developers to build high quality mobile applications easily, using their widget based SDK. The SDK, which is written in Dart, can be used to build applications that run on iOS, Android and Fuchsia. The core team is responsible for maintenance and quality control of the source code and support for the developers that work with Flutter.

8.5.2   Context Model

The figure shows a simple block-box context model implementation of the Flutter project. Here all stakeholders in the right half of the figure are part of Google Inc. Here we can see that Flutter is developed on GitHub, built in Dart and runs on three different operating systems. The three contributors that are listed are part of the core team employed by Google and have the most contributions, though there are many other contributors. All of these developers communicate via Gitter, Slack and Google Groups. The upper box shows some of the companies that use Flutter to build their own applications. Of course, there are many others working with Flutter and this set of stakeholders is just an example. Finally, in the bottom left corner the main competitors of the application are shown.

8.5.3 External Entities

Flutter is an SDK and is therefore dependent on other entities and services. These entities and services are described in the following list:

- Flutter is written in Dart.
- GitHub is used as the version control system and issue tracker.
- Flutter runs on iOS, Fuchsia and Android.
- Flutter depends on and is developed for application developers.
- Applications written in Flutter depend on the project itself.
- The developers communicate through different channels.
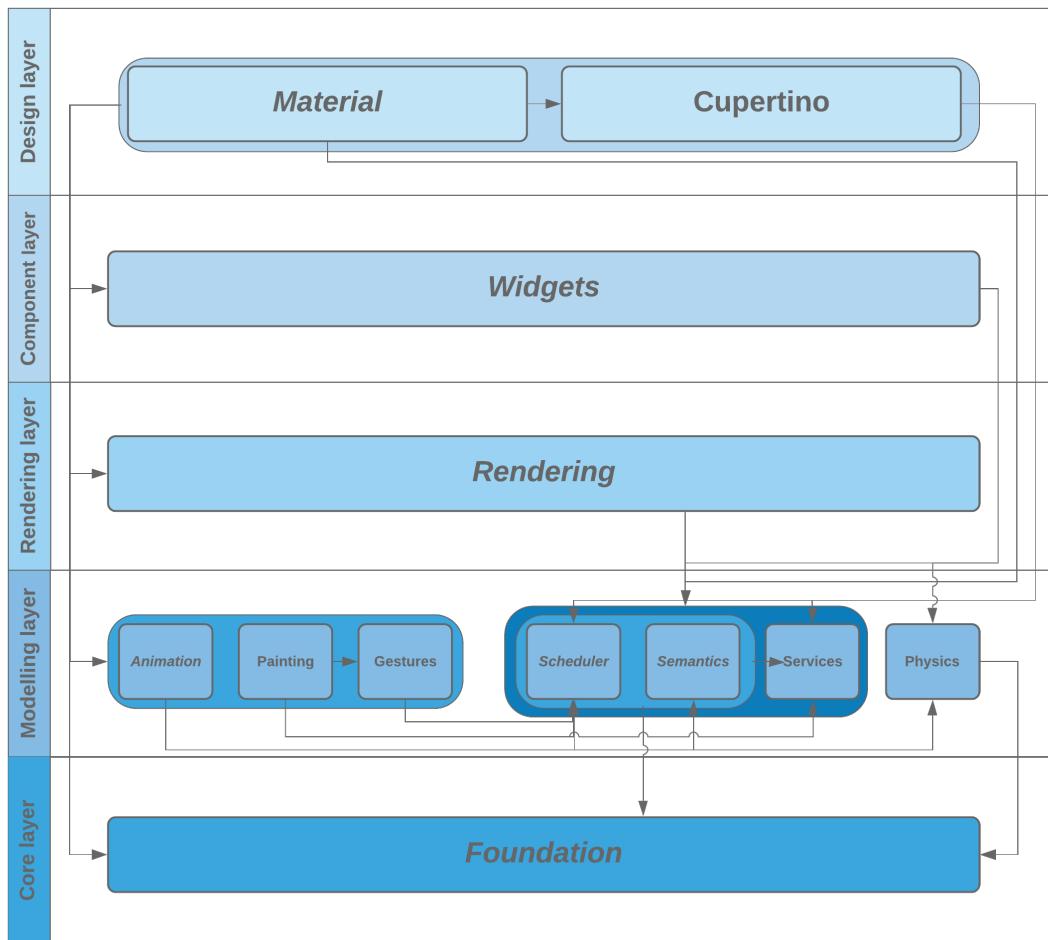
8.6   Development View

8.6.1   Module Structure Organization

In large code bases it is very important to organize source code well, to make maintenance and testing easier as well as making the architecture comprehensible for all developers working on the project. Flutter has a layered module structure, where every layer only builds on the layers underneath and on modules in its own layer. A graphical model of the module structure is depicted below. All source code of the Flutter project is divided over thirteen modules, which belong in one of the following five layers:

- The core layer, which consists of the `foundation` module, which is the base of the Flutter framework.
- The modeling layer, containing all modules that are responsible for movement and painting of elements within an application.
- The `rendering` module holds the rendering layer, which is an abstraction of the modeling layer and keeps track of which widgets belong where on the screen.
- The component layer consists of the `widgets` module, widgets are ready-to-use UI components that developers can use when building their own application.
- The design layer contains pre-defined widgets implementing the Material Design and the iOS style for the `material` and `cupertino` modules respectively.

In the graph all dependencies between the modules are drawn. The dependencies are drawn in such a way that it is easy to see which layer depends on which other layers. All the layers only depend on layers underneath (i.e., they build on each other).

8.6.2   Common Design Models

Common design models are needed to reduce duplicate effort and to increase technical coherence. In this section Flutter's standardization approaches are discussed.

8.6.2.1   Common Processing

- **Widgets**: Flutter's slogan is: "Everything is a widget". The UI is built using widgets, composed of even smaller widgets, which in turn are built using even more basic widgets. A widget can be a structural element (e.g. menu), as well as a stylistic element (e.g. padding). Widgets form a hierarchy (widget tree data structure), which is based on composition. The root node is the application itself and the leaves are the smallest widget components.

- **Rendering**: the widget building recursion tree bottoms out in `RenderObjectWidgets`, which are widgets that create nodes in the underlying render tree. The render tree is a data structure that stores the geometry of the user interface, which is computed during layout.

- **State Management**: Flutter is declarative. This means that Flutter builds its user interface to reflect the current state of your app. When the state of your app changes (e.g., a button is clicked), then the UI rebuilds from scratch.

- **Debugging**: Flutter provides the `flutter analyze` tool for static analysis. Furthermore, they provide a versatile debugger to print logs, which can be used for debugging the various application layers (widgets, rendering, scheduling, etc.).

- **Plugins**: developers can use external Dart plugins such that they do not have to build everything from scratch. The developer simply has to add the dependency to the `pubspec.yaml` file, run `flutter packages get` and finally add a corresponding import statement in their Dart code.

- **Internationalization**: if an app is deployed to users who speak another language, values like text and layouts need to be 'localized'. This can be done using the flutter_localizations package.

8.6.2.2   Standardization of Design

The Flutter developers try to adhere to the Flutter style of coding. Some of the most important design patterns and programming paradigms are:

- **Composition**: the most important design pattern is composition, the use of small objects (see *Widgets* in the previous section) with simple functionality to compose larger complex objects.
- **Declarative programming**: the developer specifies *What* should happen (and not *How* it should happen). For example: with HTML you use `<img src="./image.jpg"/>` to tell a browser to display an image and not how to load the image file or how to render the pixels.
- **Functional programming**: a prime example of functional programming is the `StatelessWidget`. These widgets do not require state and are basically just functions that map arguments to layout elements. (e.g., the `Icon` widget maps `color`, `icon`, and `size` to UI elements).
- **Generic programming**: generic typing can help developers to know how certain classes and interfaces should be implemented. Therefore, errors can be spotted early on.

8.6.2.3 Standardization of Testing

Contributors are urged to write tests accompanying their newly implemented feature. If no tests are provided by the original developer, the feature will likely regress. If they want their code to remain in the codebase, then they should include tests according to the Tests Guide. Additionally, Flutter uses Continuous Integration tools Cirrus and Chromium to automatically build and test changes submitted through pull requests.

Dart tests are written using the `flutter_test` package's API and should be included in the `test` subdirectory of the package under test. Different types of tests are supported:

- **Unit tests**: can be run using `flutter test`.
- **Golden file tests**: comparing pixels with a provided image, a so called 'golden file'.
- **System tests**: using the flutter_driver library and devicelab for testing on real Android and iOS devices.

8.6.3 Codeline models

Before contributing to Flutter, developers are invited to start by reading the general Contributing Guidelines. The Tree Hygiene document explains how to land PR's and how to request/perform code reviews. The Issue Hygiene covers how issue tracking with GitHub is done: it explains the issue labels, how prioritization/issue voting is done and how issues are assigned. Next, the very extensive Style Guide describes the guidelines concerning the actual code writing: patterns, documentation, variable naming, formatting and packages are discussed here. When making a pull requests automated build checks and test checks by Cirrus CI and the Flutter Build Bot are performed to make sure everything still works correctly.

8.6.4 Codeline organization

The structure of the system's source code is divided in multiple parts:

- `bin` The bin contains the internal engine of Flutter.

- `dev` This directory contains tools and resources that the Flutter team uses during development of the framework.

- `examples` This directory contains several examples of using Flutter.

- `packages`

  This directory contains most of the core functionality of Flutter, which is divided in `flutter`, `drivers`, `tests`, `tools`, `localization`, `goldens` and `goldens client`.

Within the `flutter` package all main functionalities are located and categorized by the following categories: `animation`, `cupertino`, `foundation`, `gestures`, `material`, `painting`, `physics`, `rendering`, `scheduler`, `semantics`, `services`, and `widgets`. The build process of Flutter is all automated through Codemagic. Codemagic verifies all builds made and creates different workflows for different application configurations for development, testing and releases. As mentioned in Standardization of Testing, Flutter uses multiple tools for continuous integration.

8.6.5   Instrumentation

Instrumentation refers to the ability to monitor or measure the level of a product's performance, to diagnose errors and to write trace information.

Flutter has several tools to help developers. One of them is the Flutter Inspector. This tool makes it easy to inspect the widget trees of an application and can be used to visualize the layout structure and diagnose layout issues. Widgets can be clicked in a running app on a device and then in the inspector, the clicked widget is selected and its field values and nearby widgets can be inspected. Also, debug paint can be added to the running application to visualize the layout constrains of all widgets.

The SDK also has the ability to test widgets. Tests can be written using a `WidgetTester`, which creates a test environment where it can build and interact with widgets and can verify that they show the correct information.

To diagnose errors, third-party crash reporting solutions can be used (e.g., Sentry or Crashlytics). These solutions provide error tracking for released apps. Since normally it is hard to track bugs that happen in apps on the devices of users, these solutions report these errors with a stack trace in a convenient dashboard where developers can see what went wrong and which devices are affected. In addition, Flutter uses the standard log system of dart by extending it into several different log components, which can all be turned on and off when needed.

8.7   Technical debt

8.7.1   Identification

Since Flutter is written in the Dart programming language we can use tools written for Dart to analyze the Flutter framework. There is a static analysis tool for code style, comments and tests embedded in the Dart SDK: the Dart Analyzer. The Flutter analyzer tool from the Flutter SDK embeds all this functionality from the Dart SDK and users of the tool should use `flutter analyze` rather than the Dart analyzer tool for their projects using Flutter. We have put focus mainly on the `flutter` package, since that is the package that will be used by other applications using Flutter.

The core team has an attitude of not working around problems but actually go and fix them. In general technical debt should always be fixed before merging new code. There are some exceptions when some (temporary) technical debt is acquired, e.g., because of a problem with an upstream dependency, disproportionate effort or because someone is already working on the solution. Technical debt in Flutter is expressed in US dollars, which is based on an average software engineer salary and the expected time it will take to solve these issues. According to Ian Hickson, at the time of writing, the 'framework has $933,000 of debt today, which isn't actually all that bad in the great scheme of things, but definitely isn't great'.

8.7.2   Code Style and Structure

As mentioned before, to check a project against errors and warnings that are specified in the Dart Language Specification `flutter analyze` can be run. For running it on the whole Flutter project, the flag `--flutter-repo` has to be added to test all the packages in the repository. This did not return any errors or

warnings. This is no surprise, since one of the requirements for landing a pull request is that the analyzer does not return any errors or warnings.

By quick inspection of the `flutter` package it seems that the code is structured well and that the SOLID principles are followed consistently. Although in a lot of widgets the `performLayout` or the `build` method are quite long (sometimes more than hundred lines). This is in some cases probably a violation of the Single Responsibility Principle. The `performLayout` method could be split into new functions which describe their single purpose.

We already mentioned that one of the primary programming paradigms used heavily in Flutter is composition. Most widgets are built this way, they are built using small objects with a small responsibility. Using them together results in more complicated widgets, e.g., the `FlatButton` widget creates a `RawMaterialButton` which consists of the given child in a `Center` in a `Container` in an `IconTheme` in a `Material` in a `ConstrainedBox` in an `_InputPadding` in a `Semantics`. All these widgets have a narrow scope of behavior, which favors the Single Responsibility Principle for these single classes. But, because of this, the indentation-based complexity becomes pretty high in some cases, especially in the sometimes very long `performLayout` and `build` methods in some widgets.

8.7.3   Testing Debt

To run all the tests and get the coverage information in a package `flutter test --coverage` can be used. Running this on the Flutter repository returns `All tests passed!` as expected. The coverage is more interesting to look at. The total line coverage of the `flutter` package is 84% at the time of writing. The core of Flutter is well-tested, subpackages like `animations`, `gestures` and `physics` have a high coverage. One problem with the current tests is that the 'golden image' tests only work on Linux. So, when someone working on the Flutter SDK runs the tests on macOS or Windows, these tests are skipped. This is also the cause of the majority of the acquired technical debt.

If we look at which lines do not have coverage, most of them are assertions in constructors, which are only executed in debug mode and thus have no influence on the final product. Also, the `material` and `cupertino` subpackages have some icon definitions which are not tested, but it would not make sense to test these since that would practically be the same as testing image files or other binary files. After inspecting some less tested Dart files, it seems that almost everything is actually pretty well tested. Most of the untested lines are some `cases` in `switch` statements and `if` or `else` branches. For improving the test coverage on these sections, more edge cases should be considered when writing tests. The edge case that seems to be untested most of the time is a device in landscape mode or if a dark theme is used instead of a light theme. Unfortunately, there are no tools for Flutter to get branch coverage.

Next to untested lines, whole functions that are untested seem to mostly be `toString`, `hashCode` and `copyWith` functions in `Widget`s and `Theme`s. These should not be functions that may break someones code. `toString` functions rely on string interpolation (e.g., `'Hi $name'`), which is a built-in functionality of Dart, so it can be argued that these functions are so trivial that these don't need tests. This argument can also be applied to `hashCode` and `copyWith`. `hashCode` uses the `hashValues` function from an external package called `sky_engine` which is part of the Flutter engine. So, assuming that the `sky_engine` package is tested well, the `hashCode` function does not need tests. `copyWith` accepts the same parameters as the constructor of the class it belongs to, and then calls the constructor with these parameters, which default to the values of the current instance of the class `copyWith` is called on.

Overall, it seems that the `flutter` package is tested pretty well. The test coverage is high on most classes, and where it is not that high, it is in most cases because of trivial functions. The project could improve a bit on test coverage by creating tests for edge cases and some locally untested functions. Mostly the technical debt can be decreased by fixing the golden tests for macOS and Windows.

8.7.4   Evolution

To see how the Flutter project has evolved over time in terms of technical debt, we will look at previous versions of Flutter. At the time of writing the latest version of Flutter on Github is v1.4.1. Since the Flutter project is relatively young, with a first release on Github not even two years ago, we have investigated and compared the current state of technical debt with the moment of Flutters first stable release (v1.0.0, Nov 2018) and the first beta release (v0.1.0, Feb 2018).

In terms of architecture, nothing much has changed since the first beta. The codebase has grown almost by 50% in terms of codelines since the first beta, but all this added code did not result in design or architectural changes. The total testing coverage has decreased a bit over time. The coverage went from 86.8% to 84.9% to 83.9% for v0.1.0, v1.0.0 and v1.4.1 respectively. However, this seems to be mostly because of added icon and constant definitions and not because of relatively less tested code.

8.8   Operational View

This section will describe how the system will be operated, administered and supported while it is running in its production environment. Flutter had been operational for about a year at the time of writing.

8.8.1   Support

Flutters support model is fully based on its open source community. This is only possible because their end users are developers and therefore also often contributors to the project. Developers are able to support each other through the different communication channels Flutter offers, giving advice to each other and having discussions on possible improvements or new features they are working on. The members of the Flutter core team participate within the community in the same way. If the platform fails completely or a severe issue occurs, the core team can be notified by creating an issue on GitHub and a high priority will be assigned to it. All members of the community are allowed to work on all issues, though the core team is responsible for the operational status of the project. As Flutter is an SDK, there is no hardware that needs any support. Therefore, only online communication suffices for customer support.

8.8.2   Operational Monitoring and Control

Operational monitoring and control is for Flutter something not to worry about, since Flutter is an SDK and there is no such thing as system down time and therefore there is no instrumentation to be found for this part. Flutter is reliable on the uptime of some third parties such as GitHub and Pub Dart, where the codebase and the Dart packages are stored respectively. However, every developer that is using Flutter can keep track of the performance of their own application with the Flutter performance profiling functionality. Within this

profiling four different parts of the application, so called 'threads', can be monitored: Platform, UI, GPU and I/O. Furthermore, for monitoring any downtime the logs can be used, which have been explained in the Instrumentation section.

8.8.3 Other Aspects of Operation

Due to the Open Source character of the Flutter project, it is fully developed on GitHub from the start. Changing from the development-only phase to the operation and development state therefore has no implications on the way Flutter is being maintained and no data transfer is needed to other environments. Everything remains fully accessible for both the developers as for the users of the SDK and nothing has to change in the way contributors develop and maintain the source code.

Another property of the Flutter project is that it is only software that is used to build other applications and it is therefore not directly coupled to any hardware installation. As a consequence, there is no need for physical installation or upgrades, configuration management (other than within the software components) or functional migration. This eases the operational phase and causes the development phase and operational phase to be highly similar.

8.9 Conclusion

This chapter analyzed Flutter, a cross platform mobile app SDK. The first section shed light on the stakeholders involved in the project. The most important ones are: the acquirer (Google), the developers (Google engineers and other members of the open source community) and the users (e.g., companies like Alibaba and other app developers). Next, from the context model, it can be seen that the most important dependencies are Dart, Fuchsia, iOS, and Android. Furthermore, some important competitors were found: React Native, Ionic, and Apache Cordova.

The next section showed an overview of the code module structure and the dependencies between these modules. The modules are structured neatly according to their respective functionalities. Next, the common design models were analyzed. It was found that there is common processing for rendering, debugging, plugins and internationalization. Furthermore, the Flutter team uses several design patterns across the code like composition by the extensive use of widgets. Flutter has an estimated technical debt of 933K USD, as indicated in an interview by Ian Hickson, one of the core team members. This was calculated by multiplying an engineer's salary with the estimated work hours left. The bulk of the debt comes from the fact that the golden file tests only run on Linux. Nonetheless, it can be concluded that the testing coverage is quite high (85% line coverage) and that the architecture and code structure are clear and strong.

In conclusion: the Flutter team, although the project is young, has very mature workflows. The code quality is impeccable, in part due to strictly enforced coding guidelines and because of the dedication of experienced engineers. Flutter is on top of its game, has clear vision and goals and will keep on competing with industry giants like Facebook's React Native.

8.10 References

1. Rozanski, N., & Woods, E. (2012). Software Systems Architecture (2nd ed.). Pearson Education.

2. Flutter documentation - https://flutter.dev/docs

3. Dart programming language - https://www.dartlang.org/

4. Flutter repository - https://github.com/flutter/flutter/

5. BSD. The 3-Clause BSD License - https://opensource.org/licenses/BSD-3-Clause

6. Flutter "Other Licenses" - https://raw.githubusercontent.com/flutter/engine/master/sky/packages/sky_engine/LICENSE

7. Flutter website - https://flutter.io/

8. One More Thing (2018-12-08). Google Flutter maakt app ontwikkeling eenvoudiger - https://www.onemorething.nl/2018/12/google-flutter-maakt-app-ontwikkeling-eenvoudiger/

9. Dieter Bohn (2019-12-04). Can Google make cross-platform mobile app development suck less? - https://www.theverge.com/2018/12/4/18125053/google-flutter-1-0-skia-mobile-app-cross-platform-developers

10. Maxdoro - https://www.maxdoro.nl/flutter/

11. It's all Widgets! - https://itsallwidgets.com/

12. Flutter Wiki - https://github.com/flutter/flutter/wiki

Chapter 9

Gutenberg

By Timo van Asten, Sven van Hal, Niek van der Laan en Daphne van Tetering.

9.1 Table of Contents

9.2 Introduction

WordPress is a Content Management System that services approximately one third of the web [1]. Mid 2016, the decision was made to start developing a new page editor experience: Gutenberg. The then-used editor – a single rich-text field – had not been updated in a decade and WordPress' competitors were rapidly catching up. Gutenberg is a visual page builder instead: all content elements are blocks and these can be arranged in any desired order. Gutenberg is shipped with WordPress since December 2018.

In this chapter, we review multiple aspects of the software architecture of Gutenberg. We first look at the bigger picture by analyzing their stakeholders and project context. Then, we dive into the inner code structure and look at the module structure, common software design, testing methods and how the codeline is organized. We continue to analyze and critique the technical and testing debt of the project, and conclude with an analysis of the often-overlooked accessibility implications of the new editor.

This chapter is part of the Delft Students on Software Architecture Book 2019.

---

[1] Usage statistics and market share of wordpress for websites, (2019). https://w3techs.com/technologies/details/cm-wordpress/all/all.

Figure 9.1: Gutenberg Logo

9.3 Stakeholders

9.3.1 Identifying stakeholders

We identified stakeholders according to the Rozanski & Woods [2] classification, as well as additional types of stakeholders that followed from our research.

9.3.1.1 Acquirers

Gutenberg is part of WordPress, which was originally founded by Matt Mullenweg. He later transferred its intellectual property to the WordPress Foundation. Mullenweg also founded Automattic, a company whose goal is to commercialize WordPress by providing a blogging platform and building custom extensions for WordPress. Mullenweg is still ultimately responsible for new WordPress releases, as well as the allocation of funds from the WordPress Foundation.

9.3.1.2 Developers and Maintainers

Since Gutenberg is an open source project, everyone is encouraged to participate in its development. At the time of writing, Gutenberg has nearly 400 contributors to the master branch. Interestingly the top seven contributors are all affiliated with Automattic.

The Gutenberg project has multiple code owners. Each of them has responsibility over certain types of files in the project, as indicated in the `CODEOWNERS` file. Three developers stand out by being code owner of (almost) the whole project:

- **@youknowriad** (Riad Benguella) is an employee of Automattic and works on WordPress. As of this writing he also is the top contributor of the project.

- **@aduth** (Andrew Duthie) is code owner of the whole project with exception of the documentation, Blocks and Widgets. He is the second largest contributor and works as JavaScript Engineer at Automattic.

- The last major code owner is **@gziolo** (Grzegorz Ziółkowski). Similar to the previous code owners he is an employee of Automattic and a major contributor to the project.

There are 13 other code owners that are responsible for parts of the project (e.g. only the documentation). These are **@chrisvanpatten**, **@nerrad**, **@Soean**, **@noisysocks**, **@talldan**, **@coderkevin**, **@jaymanpandya**, **@notnownikki**, **@nosolosw**, **@mkaz**, **@jorgefilipecosta**, **@ajitbohra** and **@mmtr**. We will elaborate on the role of some of them in other sections.

9.3.1.3 Assessors

Everyone working on the Gutenberg project is able to review pull requests on Github, which in essence makes everyone who does so an assessor. To ensure a consistency in code quality and style, several tools are in place of which some will be discussed in the Development View and Technical Debt sections.

---

[2] N. Rozanski, E. Woods, Software systems architecture: Working with stakeholders using viewpoints and perspectives, Addison-Wesley, 2011.

The code owners have a special role as assessors, since they make the final decision to merge a PR or not.

9.3.1.4   Communicators

The developers of Gutenberg hold weekly meetings on the *#core-editor* Slack channel. These meetings are chaired by **@youknowriad** and are accessible to everyone.

The Gutenberg Handbook is the main documentation for other stakeholders in this project. It is open for everyone to contribute to via Github. The quality of this documentation is assured by **@youknowriad**, **@gziolo @chrisvanpatten**, **@nosolosw**, **@notnownikki** and **@mkaz**, making them the key communicators of Gutenberg.

Other major contributors to the documentation are **@aduth**, **@dd32**, **@wpscholar**, **@adamsilverstein** and **@johnwatkins0**.

9.3.1.5   Production engineers

The Gutenberg project contains tools like package managers, linters and tools for testing which need to be configured. **@youknowriad, @gziolo, @aduth, @ntwb, @nerrad, @ajitbohra, @nosolosw** and **@mkaz** are responsible for this part of the project.

9.3.1.6   Suppliers

The following parties are suppliers for the Gutenberg project:

- Git facilitates version control.
- GitHub hosts the project and facilitates communication between the developers.
- Facebook supplies the main framework on which Gutenberg is built: React.
- Travis facilitates continuous integration.
- WordPress supplies the software on which Gutenberg runs.

9.3.1.7   Support staff

The best place to get help as a user is via the WordPress.org support forums[3], or via the Slack *#help* channel.

9.3.1.8   Testers

Testing your code is part of the PR checklist. In every PR, developers have to explain how their code was tested, which can then be verified by the other developers. Travis is used to run all tests in the project, all tests have to pass before a PR can be merged. Top contributors to the testing folder of the project include **@gziolo**, **@ellatrix**, **@youknowriad**, **@aduth** and **@dmsnell**.

To help the developers with testing, the Gutenberg Handbook has a special section on testing, which is updated once new testing procedures are in place.

---

[3]WordPress.org, WordPress.org. (n.d.). https://wordpress.org/support/plugin/gutenberg/.

9.3.1.9 Users

Anyone who uses WordPress is a potential user of Gutenberg. These users vary from individuals to some of the world's largest companies, listed here. Users may individually choose to use an alternative editor, like the classical editor, instead.

9.3.1.10 Other stakeholders

9.3.1.11 Competitors

Before the rise of Gutenberg, other page-builders were built to replace the old editor. Some of these page-builders, such as Elementor, Beaver Builder and the Page Builder by Site Origin, have more than a million downloads, making them competitors of Gutenberg. Another competitor of Gutenberg is the old editor, since users can still decide to opt-out of Gutenberg and use it instead.

9.3.1.12 Power vs. interest grid

Using Mendelow's power vs. interest grid, stakeholders can be divided into four categories. These categories are: satisfy, key stakeholders, minimal effort and inform and are shown in the figure below.

9.3.2 Integrators

According to the blog post by Georgios Gousios, integrators are responsible for 'enforcing an online discussion etiquette and for ensuring the project's longevity by on-boarding new contributors'. Within Gutenberg, most of the code owners also function as an integrator. Examples of this are @nosolosw and @youknowriad: when we posted a message in the *#core-editor* Slack channel about contributing, they replied with useful documents, like the Gutenberg Handbook. Other examples are @jasmussen and @karmatosed who are responsible for starting discussions about design and @afercia who mostly opens discussions on accessibility.

9.3.3 PR analysis

To get a clear view of the decision making strategy, we analyzed the 10 most discussed accepted and rejected PRs. From this it becomes clear that most decisions are made by having a thorough discussion on advantages and disadvantages of a certain approach. The final decision is made taking into account the scope of the pull request and the long term vision of Gutenberg. This decision is often made by one of the integrators responsible for the area that the pull request focuses on.

Within the rejected pull requests, the most occurring reason for rejection was due to the length of the discussion, during which the project itself was already developed further causing the functionality to be redundant or outdated, which required a new PR for the same functionality using a different implementation. Some pull requests were rejected because of a change in vision.

Besides analyzing PRs, we asked about the decision making process in the *#core-editor* Slack channel. @youknowriad responded that Gutenberg follows the WordPress decision making process, which is discussed
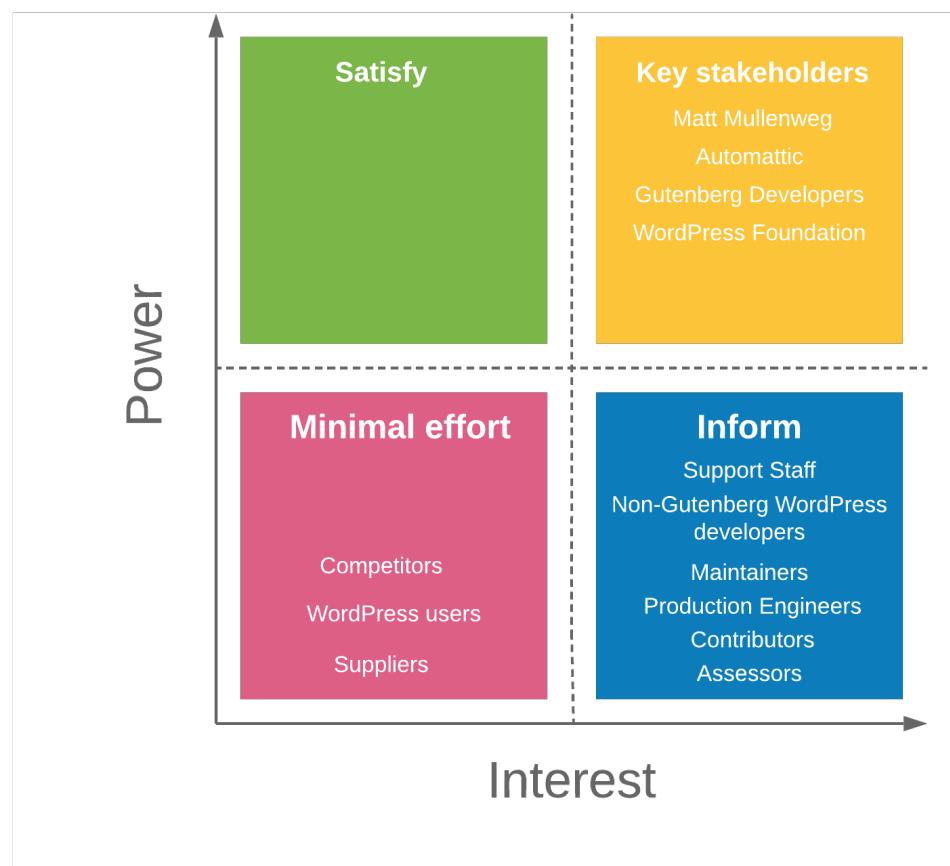
Figure 9.2: Power vs. Interest Grid

in this WordPress blog. Decisions at WordPress are made keeping the end-users in mind, who are mostly non-technically minded. WordPress states that it is the developer's duty to make smart design decisions and to avoid placing the responsibility of technical choices on end users. For example, from a developer perspective, more options on how to configure your system would always be better, because this allows for a tailored user experience. From an end-user perspective, more options result in more decisions to make on topics that most end users don't care about.

9.4 Context View

This section describes the context of the `WordPress/Gutenberg` project. A context view "describes the relationships, dependencies, and interactions between the system and its environment." [4]

The scope of Gutenberg is limited to the page editing experience within WordPress, enabling users to create complex posts and pages. Gutenberg is responsible for the interactive editing screen, converting content blocks to a suitable HTML representation and for communicating with WordPress to store posts and pages.

9.4.1 Context model
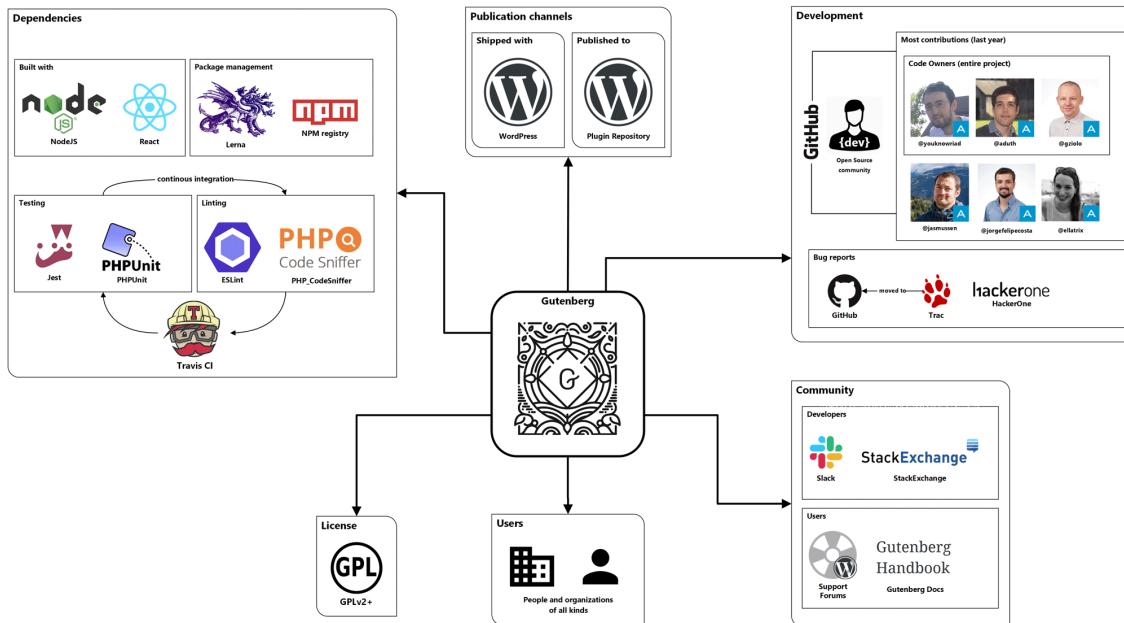
Figure 9.3: Gutenberg context model

WordPress is used by a wide variety of users and so is, by extension, Gutenberg.

---

[4]N. Rozanski, E. Woods, Software systems architecture: Working with stakeholders using viewpoints and perspectives, Addison-Wesley, 2011.

Gutenberg is written in JavaScript and built with the React framework. Additional development tooling runs on NodeJS. A small part of the codebase, the interface with WordPress, is written in PHP.

Gutenberg is developed on GitHub, where a small group of people is responsible for most of the contributions to the project. Fourteen people are *code owner*, of which three people (@youknowriad, @aduth and @gziolo) have final decision power about all parts of the project. Bug reports are collected at GitHub, and lingering bug reports at the WordPress issue tracker are moved to GitHub. Security issues can be reported via the WordPress HackerOne channel.

Gutenberg comprises multiple independent packages for different parts of the editor, which are managed through Lerna and published to the npm repository. A number of other development dependencies are also retrieved using npm.

The quality of the project is constantly monitored and a substantial part of the codebase is covered with tests. JavaScript tests are run with Jest and PHP tests with PHPUnit. The code coverage level is watched at GitHub using Codecov. Furthermore, all code is linted with either ESLint (JavaScript) or PHP_CodeSniffer (PHP) to enforce and/or fix a consistent coding style and license incompatibilities. This process is entirely automated with Travis CI.

The Gutenberg community can be divided into users and developers. Developers use Slack as their primary communication channel or ask questions at StackExchange, while users are encouraged to ask for support at the WordPress support forum. Documentation is available at a dedicated Gutenberg docs website.

Gutenberg features a faster release cycle than WordPress. Intermediate versions are published to the WordPress plug-in repository, and the most suited version is selected for a WordPress (CMS) release.

Gutenberg is licensed, as is WordPress, under the GPLv2+ license. All contributions by users also have to be under this license.

9.5   Development View

The development viewpoint is used to describe the architecture that supports the process of software development. In this section, we look at Gutenberg's architecture using the definition from Rozanski & Woods [5].

9.5.1   Module Organization

The Gutenberg repository consists mainly of packages developed for the Gutenberg editor, and uses Lerna to manage these Gutenberg modules and publish them as packages to npm.

Although the packages folder has a flat structure, we identified the responsibility of each package with respect to the logic flow of the editor. The image below, found on the Gutenberg repository, illustrates this flow.

---

[5]N. Rozanski, E. Woods, Software systems architecture: Working with stakeholders using viewpoints and perspectives, Addison-Wesley, 2011.
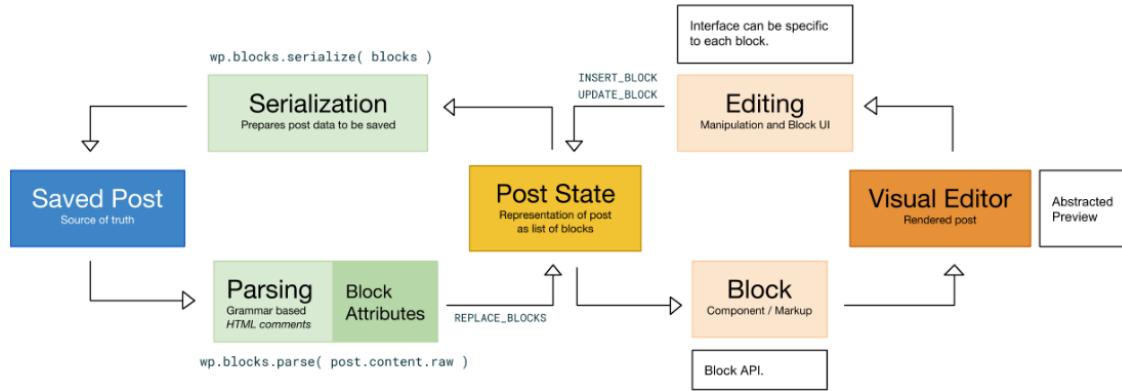
Figure 9.4: Editor Logic Flow

The editor has several responsibilities: parsing, visual representation, editing and serialization. Other packages are responsible for tooling, utilities, data management, the user interface and retrieving reusable blocks. We briefly highlight most important packages and their contributions.

9.5.1.1 Editor

The `blocks` module is responsible for converting blocks between JavaScript data objects and HTML, and is supported by the `block-serialization-spec-parser` and the `Block-serialization-default-parser` modules. The `block-editor` module is mainly responsible for the editing logic, whereas `editor` is responsible for editing non-block posts. This separation of concerns is newly introduced in Phase 2, as described in #14122.

9.5.1.2 User Interface

All blocks must be registered before they can be used in the editor. The `element` module is used to describe the structure of a block's markup. Each block can be edited by calling their edit function defined in `blocks`.

9.5.1.3 Utils & Tooling

Most packages serve as utility for the implementation of blocks, to support the editor,or simply provide reusable functionalities that can be used by developers when creating new blocks. Several tooling packages concern several responsibilities including config, plugins and testing.

The figure below displays the relations between these packages.

9.5.2 Codeline Model

The *codeline model* describes the organization of the system's source code: how is the code controlled, maintained and which (automated) tools are used in the build and release process, and what folder structure
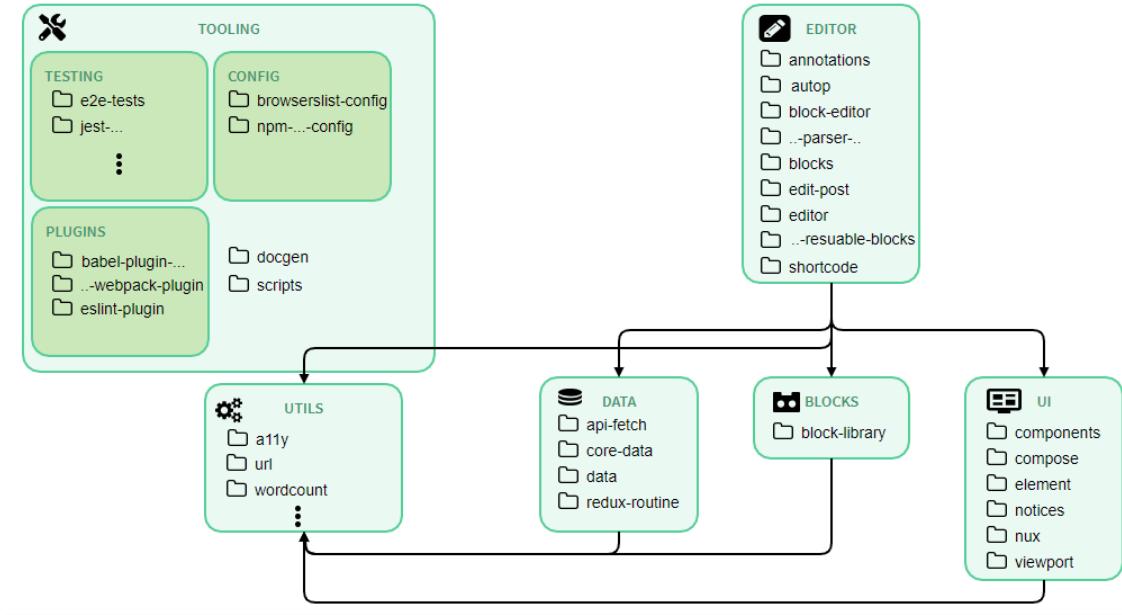
Figure 9.5: Package structure of Gutenberg

is used.

9.5.2.1   Code Management

Gutenberg uses the Feature Branch Workflow [6]. Contributors need to fork the project, make changes on a separate branch and create a pull request to the upstream master branch. Code owners of the project review the code, optionally request changes and merge the branch when the work is satisfactory [7].

9.5.2.2   Source Code Structure

The directory structure of Gutenberg is shown in the table below Every directory has a single responsibility and different types of source code based are grouped based on their purpose. The project consists of multiple modules (in the `packages` folder), which also take up the majority of the project size.

Tests are present at multiple different locations in the source code: PHP tests in the `phpunit` folder, integration tests in the `test` directory and for each package separately (optionally) a number of unit tests, as well as end-to-end tests in the `packages/e2e-test` folder.

---

[6] Atlassian, Git feature branch workflow | atlassian git tutorial, Git Feature Branch Workflow. (n.d.). https://nl.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow.

[7] Gutenberg contributor handbook, (2019). https://wordpress.org/gutenberg/handbook/contributors/.

Directory	Contains	Size	Relative
📁 assets	Project-wide (S)CSS variables and mixins	0.02 MB	0.2%
📁 bin	Installation, build and debug tooling	0.05 MB	0.7%
📁 docs	Documentation	1.78 MB	23.5%
📁 lib	WordPress PHP interface	0.05 MB	0.7%
📁 packages	Core and auxiliary JavaScript packages	4.71 MB	62.2%
└ 📁 {package_name}/src	Package source code	-	-
└ 📁 {package_name}/test	Package (unit) tests	-	-
└ 📄 {package_name}/	Package configuration files and metadata	-	-
📁 phpunit	WordPress PHP interface tests	0.01 MB	0.2%
📁 test	JavaScript (integration) tests	0.13 MB	1.7%
📄 /	Configuration files and metadata	0.82 MB	10.9%
	Total:	**7.59 MB**	**100%**

Figure 9.6: Directory Structure

9.5.2.3 Release Process

A new build of Gutenberg is released every two weeks, after having matured as a release candidate first [8]. Continuous integration is in place to automatically test and lint any code that is to be merged into the master branch and by extension a release.

Gutenberg lives primarily in the WordPress plug-in repository, but is also bundled with WordPress. The most suitable Gutenberg release is cherry-picked for each WordPress release by the WordPress maintainers.

9.5.3 Common Processing

The Gutenberg project contains many packages that handle common processing for the project. These can be divided in a few different categories:

- **Configuration:** These packages include configurations that are common throughout the project (*@wordpress/browserslist-config*) or provide a default configuration (*@wordpress/babel-preset-default*).
- **DOM, HTML and browser related processing:** Packages for manipulating or retrieving information about HTML, the DOM and the browser (*@wordpress/element, @wordpress/html-entities*).
- **Other small common processing:** Packages for small tasks like parsing date strings (*@wordpress/date*), manipulating URLs and working with keycodes.

---

[8]Gutenberg release process, (2019). https://wordpress.org/gutenberg/handbook/contributors/develop/release/.

- **Other major common processing:** This includes tasks like making API-requests *(@wordpress/api-fetch)*, internationalization *(@wordpress/wp-i18n)* and accessibility *(@wordpress/a11y)*.

9.5.3.1   Instrumentation

For logging information about the system the native JavaScript methods `console.log`, `console.error` and `console.warn` are used. To prevent temporary log statements making it into production code there is an ESLint rule in place that does not allow these log statements, except if they are explicitly allowed by disabling this rule.

9.5.4   Standardization of Design

Within Gutenberg, all content has the same internal structure called a `block`. Blocks are hierarchical units, meaning that each block can be a parent or child to another block. Within Gutenberg, it is possible to create custom blocks. To ensure the compatibility of these blocks with the parser and serializer, each block needs to have the same software design. Below, we briefly discuss the most important parts of this design, a complete overview of the internal block structure is shown in the figure below.

1. `plugin.php`: this file contains the plugin information needed to retrieve a block and list it as an available plugin in the plugin library of Gutenberg.
2. `init.php` this file is use to enqueue all JavaScript and CSS files. To achieve this, Gutenberg has two hooks: `enqueue_block_assets` on the frontend and `enqueue_block_editor_assets` on the backend.
3. `src/blocks.js` contains the JavaScript defining the behavior and functionality of the newly created block.

The new block is connected with Gutenberg using the `registerBlockType`-function from the Block API, which is called in `src/block.js`. This function takes two arguments: the block name, which has to be unique and is used to identify a block, and a block configuration object. The block configuration object contains properties such as a display title and category, these properties have to be defined to allow registration of a block.

To aid developers, several tutorials and guidelines exist, which can be found in the Gutenberg handbook. Gutenberg documents its block API and a list of block grammar containing code snippets for creating blocks. Coding guidelines are also provided to inform each developer about coding guidelines specific to Gutenberg.

9.5.5   Standardization of Testing

Tests for each package are included in the `/src/test`-folder of each package and also in `/test/`-folder in the Gutenberg project. End-to-end tests are included separately in `e2e-tests` and `e2e-test-utils` folders. To ensure code is tested sufficiently and that building succeeds, a testing overview is included in the Gutenberg repository. When testing, Gutenberg asks to take the following into account:

- The behavior(s) being tested
- The errors that will probably occur when running the new piece of code
- The correctness of the test and the risk of false positives/negatives

```
📁 new-block
  📁 dist
    📄 blocks.build.js
    📄 blocks.editor.build.css
    📄 blocks.style.build.css
  📁 src
    📁 block
      📄 block.js
      📄 editor.scss
      📄 style.scss
    📄 blocks.js
    📄 common.scss
    📄 init.php
  📄 package.json
  📄 package-lock.json
  📄 plugin.php
  📄 README.md
```

Figure 9.7: New block file structure

- The readability of the test

The following frameworks are used:

- Jest for JavaScript testing and ESLint to enforce JavaScript code style. Using `npm test` will execute the unit tests and code linting.
- Enzyme for React testing, included when running `npm test`
- Snapshot to test UI behavior, also included in `npm test`
- Google Puppeteer for end-to-end testing, run using `npm run test-e2e`
- PHPUnit for PHP testing, run using `npm run test-php`

9.6 Technical Debt

Technical debt is a metaphor that reflects overdue or additional programming work caused by choosing an easy, short-term solution instead of a better solution that would take more time. This section analyzes the technical debt of Gutenberg.

9.6.1 Presence of Technical Debt

We used SonarQube to detect code smells and potential bugs. Only the `lib/` and `packages/` directories have been considered, to prevent analyzing test code and tooling.

When using SonarQube's default quality profile, a technical debt of 10 *man-days* (80 hours) is reported, representing only 5% of the total project size. Manual inspection of the detected code smells reveals that the most frequently violated rules are violated because of common practice and design choices and do not present actual technical debt. An example of this is shown in the figure below.

> [JavaScript] Define a constant instead of duplicating this literal "*string*" 3 times.
>
> [SCSS] Unexpected duplicate selector "*selector*"

Figure 9.8: Example of violated rules due to design decisions

Since the default quality profile is very strict and it penalizes very common practices, we decided to exclude the rules in the figure from the profile. This result in a technical debt of only **7** *man-days*.

Other clear examples of technical debt are empty `if`-statements (accompanied by a TODO-statement), functioning as a scaffold to be filled in later, dead code after a wrong refactoring or duplicate CSS rules. Efforts have been made to reduce the latter in #14520 and #14546.

9.6.1.1   Copy-paste code

While writing this document, the Gutenberg team transitioned to Phase 2, which is centered around expanding Gutenberg from the editor the Customizer, the part of WordPress that allows users to create their own content. Resulting from this is a large refactor, moving components from the `editor` package to a package called `block-editor`, an example of this is seen in PR #14420. A lot of duplicate code results from this, increasing the risk for the Shotgun Surgery code smell. Running JSInspect yields the following result: `21 matches found across 874 files`, mostly related to the earlier mentioned refactor. These problems will likely be resolved in the near future.

9.6.1.2   Comment inspection

One indicator of technical debt are TODO comments. At the time of writing, the Gutenberg project has a total of 47 TODOs in the code base. These TODOs can be categorized in the following categories:

Type	Occurrences	Example from the code base
Needed improvements	21	// TODO: Search input should be focused immediately. It shouldn't be necessary to have 'waitForFunction'.
Suggested improvements	8	// TODO: Future enhancement to add an upload indicator.
Removal reminders	6	// TODO: The following is for back-compat with WP 4.9, not needed in WP 5.0. Remove it after the release.

Type	Occurrences	Example from the code base
Research reminders	5	// TODO: Figure out a way to generate docs for dynamic actions/selectors.
Thoughts and questions	4	// TODO: Should we differentiate BACKSPACE and DELETE?
Refactor reminders	3	// TODO: Refactor click detection to use blur to stop propagation.

Many of these TODOs also give some indication on why the TODO was introduced, some of which are not valid anymore. An example of this is shown in the figure below.

```
// TODO: This test should be enabled once
issue #7458 is fixed.
```

Figure 9.9: Example of an invalid TODO comment

When looking at this issue, we see that the issue itself was closed in October 2018, but the TODO is still present in the code base, which indicates that these TODOs can linger for quite a while. This is shown in the figure below.
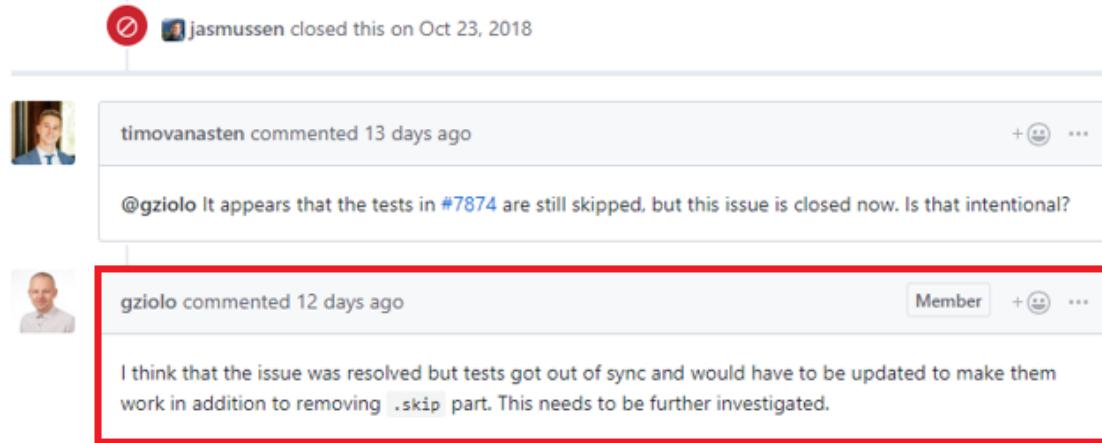
Figure 9.10: Example of a lingering TODO

9.6.2   Testing debt

To identify testing debt, issues labeled Needs Tests were reviewed. Even though this label is only used on 4/1630 open issues, they have been open for quite a while. The `--coverage` option in Jest was used to generate a coverage report for each folder, file and line. This shows that the code coverage of Gutenberg is about 50%. A snapshot of this webpage is shown in the figure below.

All files

55.59% Statements 9334/16791   **49.91%** Branches 4969/9956   **50.88%** Functions 2209/4342   **55.52%** Lines 9124/16433

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File		Statements		Branches		Functions		Lines	
gutenberg		100%	3/3	100%	0/0	100%	1/1	100%	3/3
gutenberg/packages/a11y/src		100%	40/40	85.71%	12/14	100%	5/5	100%	39/39
gutenberg/packages/annotations/src/store		96%	24/25	88.89%	16/18	100%	9/9	96%	24/25
gutenberg/packages/api-fetch/src		97.96%	48/49	96%	24/25	90.91%	10/11	97.92%	47/48
gutenberg/packages/api-fetch/src/middlewares		89.8%	88/98	77.94%	53/68	85%	17/20	89.36%	84/94
gutenberg/packages/autop/src		83.54%	132/158	70.91%	39/55	53.33%	8/15	83.66%	128/153
gutenberg/packages/babel-plugin-import-jsx-pragma		94.74%	18/19	87.5%	7/8	100%	4/4	94.74%	18/19
gutenberg/packages/babel-plugin-makepot/src		39.8%	39/98	37.5%	21/56	42.86%	6/14	39.58%	38/96
gutenberg/packages/babel-preset-default		85.71%	24/28	66.67%	8/12	100%	6/6	85.19%	23/27
gutenberg/packages/blob/src		41.67%	5/12	66.67%	4/6	25%	1/4	41.67%	5/12

Figure 9.11: Snapshot of the HTML-page generated by Jest

Issue #13812 shows that the developers are aware of the need for tests. Since this issue was opened recently, it also shows that the developers are trying to improve coverage. However, PR #14420 shows one of the reasons why the test coverage of Gutenberg is fairly low: two code owners, @aduth and @youknowriad decide to merge a PR with failing end-to-end test. This is visible in the figure below.
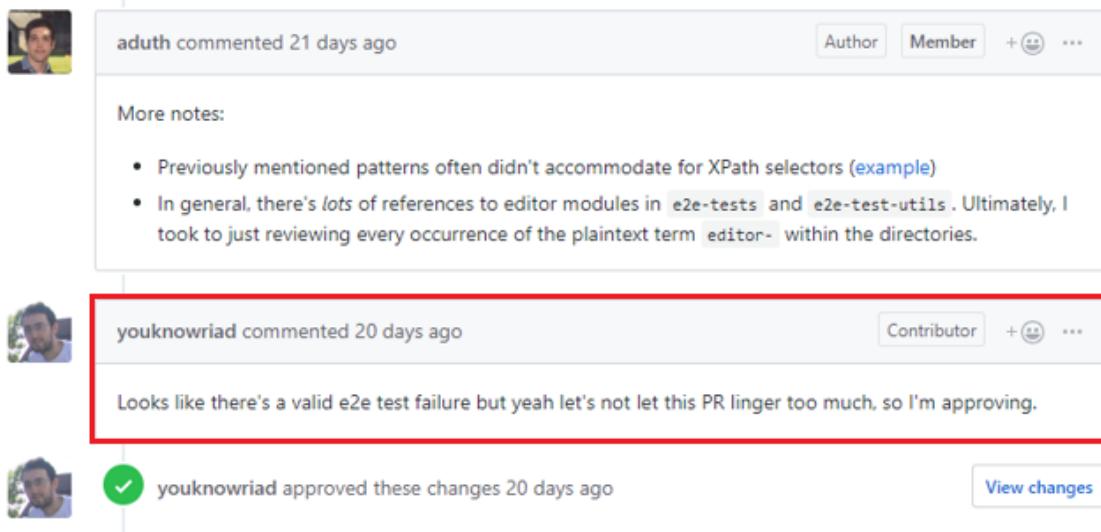
Figure 9.12: Example of a decision causing testing debt

9.6.3 Technical debt mitigation

To mitigate technical debt, Gutenberg employs a consistent workflow starting from the creation of an issue of GitHub. When this issue is picked by a developer he or she is guided by the extensive contributing guide including documentation on testing, workflow and package management [9].

During development, technical debt is kept at a minimum by running several linters before allowing a developer to commit. Examples of these linters are JavaScript, JSON and PHP linters.

After resolving an issue or when in need of help, a pull request (PR) is opened. Each PR has to include the checklist shown below:

- My code is tested.
- My code follows the WordPress code style.
- My code follows the accessibility standards.
- My code has proper inline documentation.
- I've included developer documentation if appropriate.

In addition to automated tests, each pull request is reviewed manually by at least one of the code owners. Additionally, PRs can only be merged with approval of at least one code owner.

Finally, PRs can only be merged with approval of at least one code owner. A code owner always checks whether these requirements are met before merging the PR.

9.6.4 Impact and evolution of Technical Debt

In general, the impact of the technical debt of Gutenberg is very limited, which is remarkable for a project that grows and evolves so quickly. When using SonarQube to analyze the five previous releases, it becomes clear that with newer releases the amount of technical debt only increase slightly, even though the project grows significantly. It is also visible that the number of code smells and the number of bugs that are introduced with every release is relatively minimal.

However, the rather large testing debt of Gutenberg is one of their major issues, which reduces maintainability and can cause undetected broken functionality with each code change. Furthermore, the various TODO comments in the code and empty if statements make for a dirty code base, possibly discouraging contributors to provide clean code themselves. Additionally, the large amount of duplicate code resulting from the `editor` refactor and the duplicate CSS rules hinder the maintainability of the project, as required changes are cascaded to many parts of the system. The amount of todo's and abandoned tests as a result of rushed pull requests are ramping up, putting Gutenberg at risk of this debt getting out of control.

9.7 Accessibility Perspective

Software accessibility refers to the ability of users – regardless of any disability or impairment – to (fully) use the software. In order to create and maintain a WordPress website, people suffering from disabilities – ranging from e.g. motor impairments to (color) blindness – are dependent on an accessible editor. Because

---

[9]Gutenberg contributor handbook, (2019). https://wordpress.org/gutenberg/handbook/contributors/.

they may be unable to operate certain hardware like a mouse or use assistive technologies like screen readers, software has to adapt to this.

Although accessibility is required by just a small percentage of users, the impact of inaccessibity is significant. Shipping an inaccessible editor might cost someone's job, if that job depends on using WordPress with Gutenberg. Accessibility is also a legal requirement when software receives government funding [10].

The Web Content Accessibility Guidelines (WCAG) is the most important collection of (web) accessibility standards. WCAG has a number of levels, of which "Level AA", and is commonly enforced by governments [11]. WordPress has also committed to using WCAG Level 2.0, stating that:

> All new or updated code released in WordPress must conform with the WCAG 2.0 guidelines at level AA. [12]

In the remainder of this chapter, we show that these guidelines are violated by Gutenberg regardless. We discuss why this happened, how this could have been prevented and what can be done in order to improve software accessibility.

9.7.1   Interview with Rian Rietveld

Rian Rietveld is the former WordPress Accessibility team lead. She resigned due to codebase and political issues with Gutenberg. We contacted her for an interview to understand how this happened.

She cited three main issues that caused Gutenberg's poor accessability:

- The accessibility team had communication issues due to a **knowledge gap**. On the one hand you had the developers who did not know enough about accessibility to make the software accessible, and on the other hand the accessiblity team were not knowledgable enough in React to implement improvements themselves. Rian thinks this problem is partially caused by universities not putting enough focus on accessibiliy in their education programs.

- **Accessiblity was an afterthought** in the project:

  > "We were kept at a distance by the developers until a big part of the project was already done. What was then presented turned out to have big accessibility issues. Making a project like this fully accessible in a late stage of development is almost undoable"

- **The leadership of WordPress did not care enough about accessiblity**, causing Automattic developers to leave accessibilty issues to be fixed by volunteers, who cannot keep up with the pace of paid developers. Furthermore, Matt Mullenweg unexpectedly set a hard deadline for Gutenberg to be shipped as the main editor, while the software was not nearly accessible enough at that point.

According to Rian the new Gutenberg editor is a big step back compared to the old classic editor: > "If you replace Gutenberg with the old editor, WordPress is the most accessable CMS right now."

---

[10]Designing software that is accessible to individuals with disabilities, (2019). https://www.washington.edu/doit/designing-software-accessible-individuals-disabilities.

[11]Making your service accessible: An introduction, GOV.UK. (n.d.). https://www.gov.uk/service-manual/helping-people-to-use-your-service/making-your-service-accessible-an-introduction.

[12]WordPress coding standards, (2019). https://make.wordpress.org/core/handbook/best-practices/coding-standards/accessibility-coding-standards/.

As one of the biggest issues she mentioned the fact that the sidebar, containing all options of the editor, is completely unusable using just a keyboard, because blocks are deselected when navigating to the sidebar. By conducting the Gutenberg acccessibility test ourselves, we verified that this issue is still not resolved months after her resignation. This test also revealed two other accessibility issues which we reported on GitHub: #14753 and #14754.

9.7.2 Current Situation

9.7.2.1 Workflow

Since the Gutenberg development team is rather small, the core-WordPress accessibility team is responsible for accessibility testing and reviewing for Gutenberg. To ask the WordPress-core team for a review or advice, the Gutenberg repository contains labels `Needs Accessibility Feedback` and `Accessibility` used for issues and pull-requests. The core-team discusses these issues and PRs during their weekly bug-scrub and/or team meeting on Slack, where they have their own #accessibility-channel. The discussions are often led by @afercia, who is committed to keep WordPress accessible and often reviews the Gutenberg PRs and issues. For each milestone, the team conducts a ticket triage to keep track of important issues.

9.7.2.2 Testing

Currently, the WordPress accessibility team is working on automated tests in the wp-theme-auditor repository, which contains the Deque Labs' axe accessibility testing framework. They also have opened a call for testing, asking the WordPress community for feedback to make the tool as useful as possible.

To gather user test data specifically to Gutenberg, the accessibility team has created a step-by-step testing plan, in which they ask users to perform a number of tasks and report their findings.

9.7.2.3 Support

To support developers and users, the core Accessibility team has its own webpage containing information, where all resources regarding accessibility can be found, examples of this are the Accessibility Coding Standards, Accessibility Handbook and the Accessibility Forum can be found.

9.7.2.4 Assistive Technologies

A lot of work is being put into improving Gutenberg's accessibility [13]. Currently, the following assistive technologies are present:

1. Keyboard navigation: Gutenberg includes mechanisms for navigating through the editor, traversing blocks and other important actions while only using the keyboard.
2. Screen navigation: Gutenberg includes several drop down menus which enable the user to navigate through blocks without using the keyboard.

---

[13]Regarding accessibility in gutenberg, (2018). https://make.wordpress.org/core/2018/10/18/regarding-accessibility-in-gutenberg/.

3. Audible messages: the WordPress accessibility package contains a speak function which announces edits for users with visual impairments. To improve the readability of the Gutenberg code by a screen reader, ARIA landmarks are used. These and other used landmarks can be found here.

4. High-contrast mode: the high-contrast modes helps users with vision issues by only using high-contrast colors and a black background.

5. User accessible content: an essential part of the Gutenberg editor is the ability for users to create their own content. To ensure the accessibility of the content, the guidelines from the handbook are included in the editor, whenever a user (unintentionally) violates these rules a warning is shown to alert the user.

To ensure a consistency in these technologies throughout the whole WordPress platform, each JavaScript component with built-in accessibility mechanisms is published in NPM to encourage and allow the reuse of these components. Besides this, WordPress also released a plugin to help with the most common accessibility issues.

9.7.3   Pressing Issues

Two topics are repeatedly addressed during accessibility meetings: 1) updating the User Handbook and 2) automated accessibility testing. There had passed weeks without any update on the user handbook regarding documentation on accessibility for developers working on the Gutenberg Project. However, in a recent meeting, it was announced a draft would be submitted within a week (https://make.wordpress.org/accessibility/2019/03/29/accessibility-team-meeting-notes-for-3-29-2019/). Automated accessibility testing is currently being addressed with regards to WordPress themes [14], but is still not addressed for Gutenberg specifically.

All issues regarding accessibility are categorized by priority in the ticket triage, ranging from 'normal' to 'high' to 'omgbbq'. From this, we can see that keyboard navigation related issues are currently prioritized. As also stated in the accessibility meeting on 15 March, the issue #13663 related to improving keyboard navigation between the block inspector and the block content, is one of the keystone items they want addressed [15]. Prioritization around focus management specifically regards focus management around the block toolbar (#6165, #6336, #11774).

Conclusively, they are anticipating that the WP Campus accessibility audit of Gutenberg will be delivered soon, and may contain important information for them to absorb [16].

9.8   Conclusion

Over the course of this chapter, we analyzed various aspects of WordPress' core editor Gutenberg. Having been rapidly developed and shipped to power over one third of the web only last December, Gutenberg has already established itself as a solid page building platform.

---

[14]Call for testing: Wp-theme-auditor, (2019). https://make.wordpress.org/accessibility/2019/03/26/call-for-testing-wp-theme-auditor/.

[15]Accessibility team meeting notes: 3/15/2019, (2019). https://make.wordpress.org/accessibility/2019/03/21/accessibility-team-meeting-notes-3-15-2019/.

[16]Accessibility team meeting notes: 3/15/2019, (2019). https://make.wordpress.org/accessibility/2019/03/26/accessibility-team-meeting-notes-3-22-2019/.

The most important stakeholders are the WordPress Foundation and Automattic. Matt Mullenweg, the founder of WordPress, is a key figure behind these organizations. The project exists in a rich ecosystem of open source tooling and devoted contributors.

Gutenberg consists of many different, independent packages which have distinct responsibilities. This reduces code duplication and promotes re-use of existing packages, but managing these packages can be challenging.

Taking into consideration the rapid development of the project, Gutenberg has a relatively low amount of technical debt. However, some debt is significant or has been lingering for a long time, which needs more focus from the core team.

Lastly, the accessibility of Gutenberg was analyzed. We interviewed Rian Rietveld, former WordPress Accessibility Team lead, and found that accessibility was often overlooked during development and only fixed as an afterthought. Because accessibility is essential to so many of us, we feel that not only Gutenberg has to focus more strongly on this, but software developers in general.

Chapter 10

Home Assistant

By Obinna Agba, Gijs Reichert, Mark Schrauwen, Zhiyue Zhang

10.1 Table of Contents

10.2 Introduction

More and more devices are being created with internet—aware capabilities. These devices, collectively called Internet of Things (IoT), allow for remote monitoring and control. However, there is currently no agreed upon standard for connecting these devices and controlling them from a unified interface. This is the problem that Home Assistant (**HA**) seeks to solve. HA aims to be a controller hub where users can monitor and control all the IoT devices in their houses. It also allows for integration of external data services to augment the capabilities of the connected devices and improve the overall user experience. What began as a simple script by founder Paulus Schoutsen to control a few smart devices in his house has over the last five years grown into one of the most popular IoT hub solutions today and continues to grow. The rich open source community involved in this project and the pervasive nature of IoT devices make HA an interesting choice for this analysis.

10.3   Stakeholders

An in depth analysis of the stakeholders (including Pull Requests) can be found in the appendix.

This section provides an overview of the various stakeholders that are involved with the HA project. The roles, power and interest of different stakeholders have been analyzed and summarized in the section below.

> A stakeholder in the architecture of a system is an individual, team, organization, or classes thereof, having an interest in the realization of the system. [1]

10.3.1   Rozanski and Woods classification

Using the book "Software Systems Architecture" of Rozanski and Woods [1] the HA project was analyzed to determine the stakeholders that apply to this project.

Table 1: stakeholder overview

Stakeholders	Description
Acquirers	Acquirers are stakeholders that oversee the procurement of the system. The company Nabu Casa provides project funding. Also, Ubiquiti Networks has hired Paulus Schoutsen to reach the HA goals.
Assessors	Although there is no clear assessor role for one person or team, there is a clear structure to ensure conformance to standards, legal regulations and the **privacy policy**. Every contributor has to sign the **Contributor License Agreement** (CLA) which is based on the Apache 2.0 license. [2] A github bot makes sure that the tag *cla-signed* or *cla-not-signed* is attached to each pull request to ensure the CLA is always signed before it can be merged. Every pull request is then checked by relevant code owners and/or important core contributors/maintainers.
Communicators	The most important communicators are the **core team members**, Discord moderators and the **documentation**. Of the core team members, founder Paulus Schouten [@balloob](https://github.com/balloob)) is an important communicator who spoke at a **state of the union meetup** for HA which was hosted by ING in Amsterdam.
Users	The users of HA are mostly people with technical and/or programming background with an interest in automating their homes.

Stakeholders	Description
Testers	Testing is done using unit and integration tests which are triggered to run in multiple different configurations on the Continuous Integration platform Travis CI. After each run of tests, the tool Coveralls is called to check the code coverage. The unit and integration tests are written by various developers contributing to HA.
Developers	The developers of HA are all the people contributing to the repositories. There is a core team of developers and a large group of users that contribute code.
Support staff	The support staff consists of the Github community and the Discord Server where you can ask questions and receive support from the community.
Suppliers	Some of the most important suppliers are Github, Python, Docker and Travis. These suppliers make sure the project can be developed, maintained, tested and used. There are more suppliers that enable HA to function.
Maintainers	Most components and platforms in the HA project have an associated **codeowner**. Whenever a piece of code is touched that someone "owns", they will be called to review the pull request. This way the person with the most experience with that piece of code will always be called as a reviewer. These code owners together with other core developers such as founder [@balloob](https://github.com/balloob) can be regarded as the maintainers of HA.

Table 2: extra stakeholders

Stakeholders	Description
Associated Projects	Different GitHub projects are associated with HA. Home-assistant.io which builds the documentation for the HA project and home-assitant-polymer which builds the fronted for HA. Another project, hassio, builds the first private cloud supervisor for home automation.
End-users	Users of HA with only an interest in using the HA (not developing).

Stakeholders	Description
Translators	This project is translated in at least 49 languages with many translators supporting the project and translating new strings and components.

10.3.2   Power versus Interest graph (PI Graph)

The Power/Interest (PI) graph in Figure 1 is used to visualize the PI relationship for the **stakeholders** as well as other entities of the context model. It was introduced by Gardner et al. [2] to visualize the relative importance of stakeholders.

10.3.2.0.1   Developers   HA is highly influenced by the core developers. They are largely responsible for which new features are added. Also, some members of the core team earn income (see Context View) from their work on the project, hence the high interest and power.

10.3.2.0.2   Components   HA primarily aims to be a control hub for IoT devices. As a result, its efficacy is largely dependent on the number of devices and services which it can integrate. The protocols exposed by these devices hence have a large influence on the architecture of HA. However, manufactures of these devices/service do not consider the effects of their design and implementation choices on HA's architecture/design.

10.3.2.0.3   Competitors   Competitors of HA would typically have a relative low interest in HA. Also, and especially because of HA's non-commercial nature — activities of its competitors do not affect the direction of the project as a whole.

10.3.2.0.4   Funding   Funders have a high interest in the project hence their decision to fund but they have limited influence on the project's direction. This low influence is due to the decision of the funders to allow the core developers maintain autonomy on the project. Although the amount of influence funders have is debatable, the true open-source nature of the project will most probably mean the project will suffer from funding being cut at first but will eventually find a way to continue.

10.3.2.0.5   Other PI Groups   The sponsors can be considered to be in between quadrants. Although the platforms provided by the sponsors are important, they do not have that much power. Whenever they decide to quit or influence the project the developers can decide to move to another platform. Of course, switching is most of the time undesirable as it is time-consuming and requires changes. However, when a sponsor would leave or demand changes there will most likely always be an alternative.

10.4   Contact persons

The HA project is maintained by a group of core developers from which the following are potential contact persons:

1. Paulus Schoutsen ([@balloob](https://github.com/balloob)) creator and core developer and contact person for this project.
2. Fabian Affolter ([@fabaff](https://github.com/fabaff)) core developer.

Based on [4] and [5].
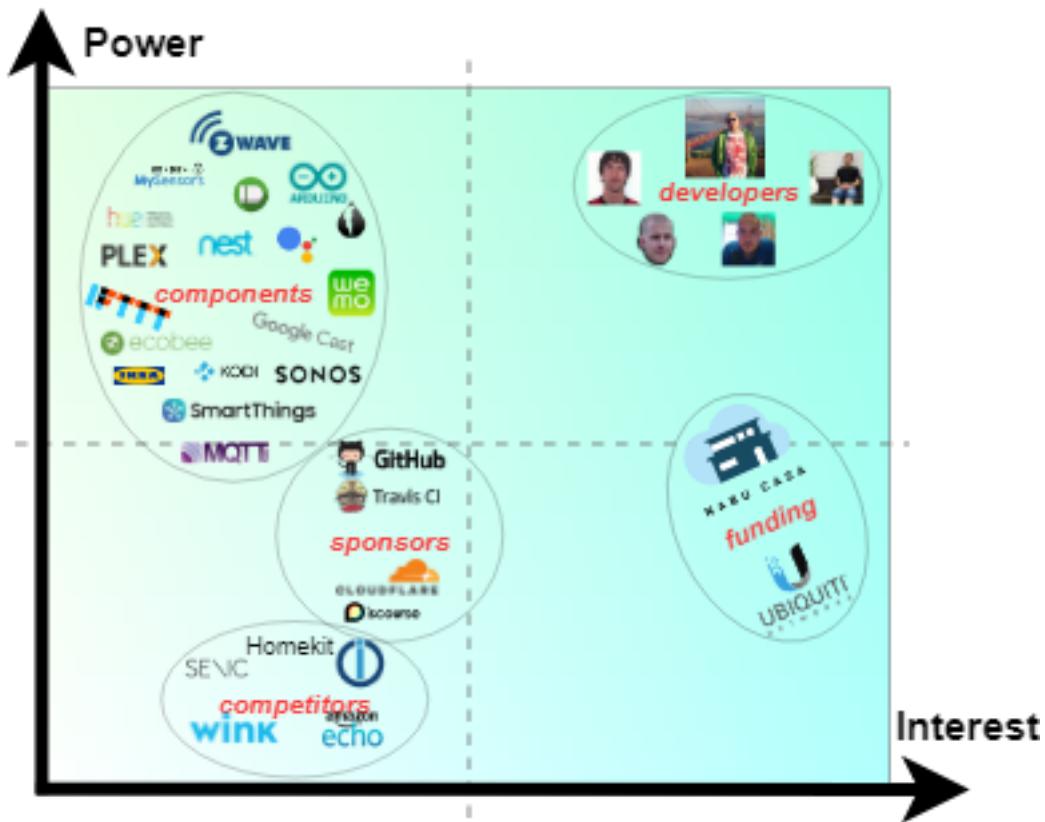
Figure 10.1: The Power-Interest Graph

Figure 1: The Power-Interest Graph

10.5   Context View

Rozanki and Woods in their book *Software Systems Architecture* define the context view as follows: > The Context View of a system defines the relationships, dependencies and the interactions between the system and its environment i.e the people, systems and external entities with which it interacts.

This analysis adheres closely to the definition put forward by Rozanki and Woods. The following sections give a detailed view of the various components which comprise the context view.

10.5.1   System Scope and Responsibilities

At the core of HA are the IoT devices. In recent years, the number of IoT devices has increased but there has not been any formal standard for the interconnection of these devices or some central control system for all devices. HA seeks to solve this by providing the following:

- A central hub which displays device state in real time
- Options to control connected devices.
- Allow the definition of event based triggers for devices.
- Allow for automated home control while providing a manual override option.
- Running locally instead of the cloud.
- Blend in with existing IoT control workflows as opposed to replacing them.

This scope definition provides a guideline for the development of HA.

10.5.2   External Entities

HA being a control hub means requires that it interacts with many external platforms/systems. These external systems are classified in three high-level categories:

1. Development Systems
2. Deployment Systems
3. Integration Systems

10.5.2.1   Development Systems

These refer to the systems with which HA is built.

10.5.2.1.1   Programming Language(s)   HA as a whole is made up of two parts: * HA Backend * HA Frontend

The backend is developed using *Python* (version 3.5 and above). The frontend is developed using Javascript, CSS and HTML. More specifically, the frontend uses *Polymer* a reusable web components library.

10.5.2.1.2 Testing HA uses *Tox* for automating and running unit tests in the backend. The frontend utilizes *Mocha* which is a Javascript testing framework. Standardization of Testing provides a more detailed description regarding testing.

10.5.2.1.3 Source Control and Issue Management HA uses *Github* for source control management. It makes extensive use of Github's issue tracking system for managing issues in the project and Github's pull request feature with which issue fixes and development of new features are managed.

10.5.2.1.4 Continuous Integration HA has a detailed continuous integration pipeline for automating the build and testing process. For the backend, this pipeline involves the following:

- Hound which is used to check for violations against the project's style guidelines.
- Travis CI which is used to run the projects test.
- Coveralls which is used to track the test coverage of the project

For the frontend, the continuous integration pipeline involves the following:

- Travis CI which is used to run the projects test.
- Netlify which is used to generate a running preview of the front end

10.5.2.2 Deployment Systems

This refers to the platform(s) where HA software can run on. HA can run on most linux distributions but the authors of the project recommend deploying on a Raspberry PI. In addition, they provide Hass.io which is a system based on a custom operating system HassOs. Hass.io provides an easy to use solution for setting up and managing HA on a Raspberry Pi.

10.5.2.3 Integration Systems

These refer to all the devices, systems, interfaces which HA aims to control or track. These entities are a major part of HA as the project's value is dependent on the quality and quantity of the integrations it provides. At the time of writing, there are 1310 of such entities. These entities (or components in HA lingo) are subdivided into the following categories:

- Air Quality
- Alarm Control Panel
- Binary Sensor
- Climate
- Cover
- Fan
- Light
- Lock
- Media Player
- Remote
- Sensor
- Switch

- Vacuum
- Water Heater
- Weather

While the above classification relies on functionality as a distinctive characteristic, it is also possible to classify these integrations based on the interface to HA. Examples of grouping based on interface would be: connection via local wireless network (e.g remote door bell), control via the cloud (e.g Amazon Echo), communication protocols (e.g MQTT, ZWave etc).

10.5.3 Funding and Sponsors

For many open source projects funding is often a pain point. There are currently two prominent sources of funding for the HA project:

1. Ubiquiti Networks

Ubiquiti Networks hired HA founder Paulus Schoutsen to work on HA. This allowed him to work full-time on the project.

2. HA Cloud (Nabu Casa Inc)

Paulus Schoutsen together with Pascal Vizeli and Ben Bangert, also key contributors, started Nabu Casa Inc to provide a subscription based cloud extension of HA. The team of Nabu Casa still contribute directly to HA and the revenue from the subscription service helps fund their work on HA as well.

10.5.4 Sponsors

HA receives sponsorship mainly in the form of free subscriptions to services from the following platforms:

1. Github
2. TravisCI
3. CloudFlare
4. Discourse

10.5.5 Competitors

HA is not the only player in the IoT market which aims to be a central control hub for all devices. There are two kinds of competitors with HA: Direct and Indirect competitors

10.5.5.1 Direct Competitors

These competitors perform a similar function and can be exact HA replacements. Examples of these include:

- Wink Hub
- IoBroker
- Senic

10.5.5.2 Indirect Competitors

These competitors can be used instead of HA but are not exact replacements. Also, the HA team provides integrations to these *Indirect* competitors which means that they can coexisit with HA. Examples of these include:

- Amazon Echo
- Apple Homekit
- Google Home

10.5.6 Community

HA has a vibrant community both of developers and users. The following platforms are used by the HA Team/Community.

- Github This is used to manage the code for the various components of HA
- Discord This provides a more informal and supportive platform where users can get help with problems with running HA
- Discourse This is a forum for have discussion around HA including its development and usage.
- Blog This is mainly used as an announcement forum for updates to HA and notable events. Also, tutorials on specific HA use cases are sometimes shared.
- Hass Podcast The HA team also runs a podcast where HA related topics are discussed.

10.5.7 Context Model

The context view describes the scope, responsibilities, relationships, dependencies and interactions around a project. A visual depiction of these different components of the context view is given in Figure 2 below.

Figure 2: Context Model

10.6 Technical Debt

Technical Debt (**TD**) can be defined as the extra time needed to implement a change when code parts are not optimally implemented. If a project suffers from, it is more expensive to maintain.

10.6.0.1 Truck Factor

The truck factor is defined as the minimum number of project members who need to leave for it to become inactive. To determine this, first, consistent contributors (> 20 commits per year) since the project's inception were counted. The results of this analysis is shown below:
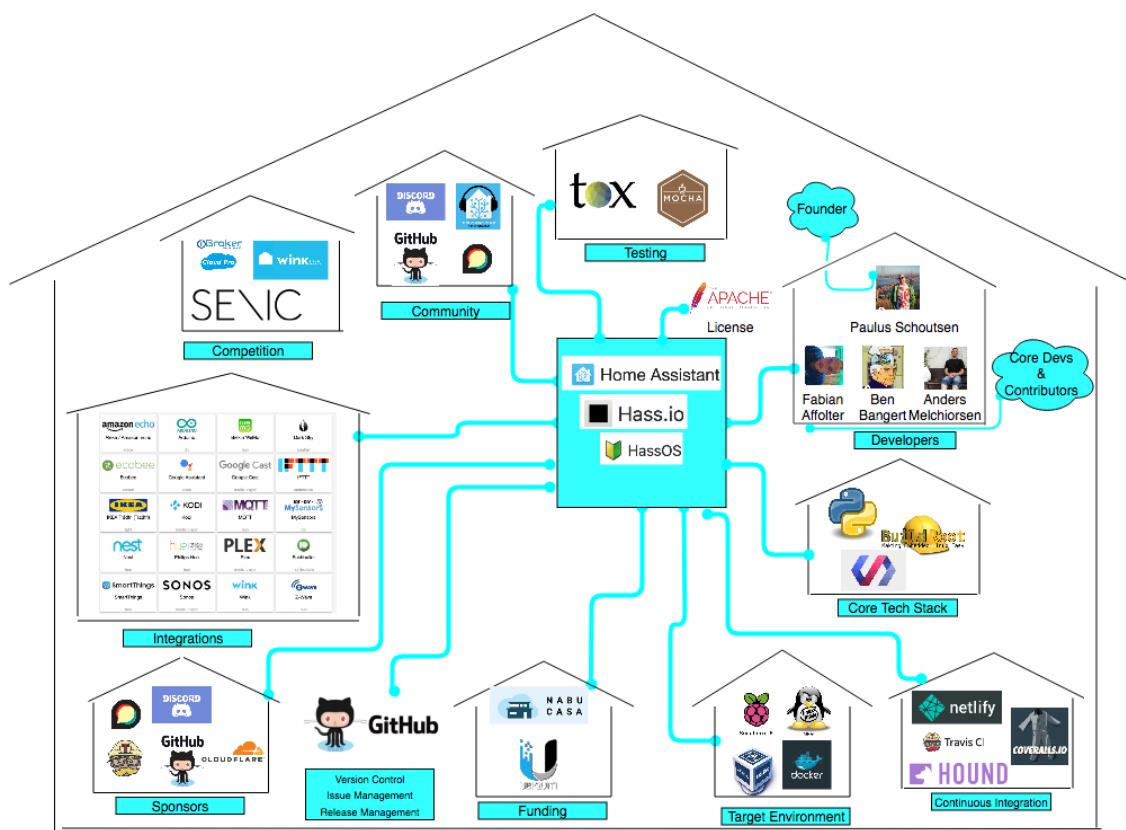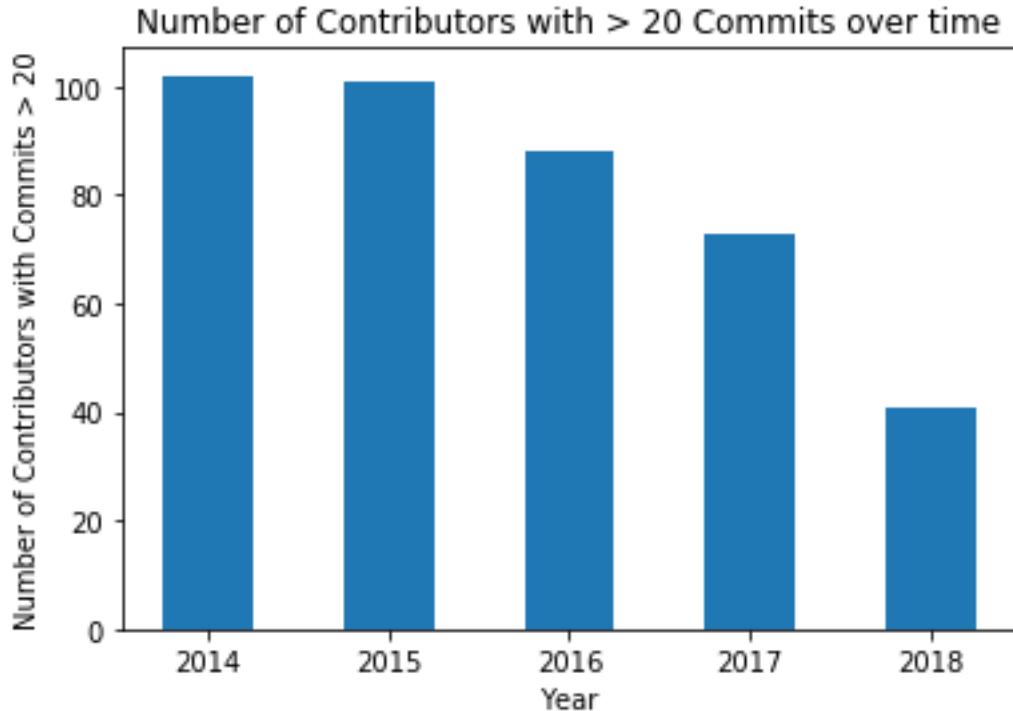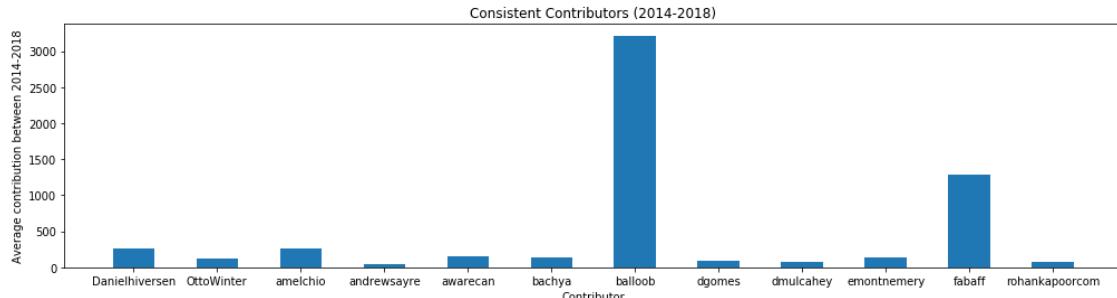
Figure 10.2: The Context Model

Figure 3: Number of contributors with more than 20 commits

A better indicator of the truck factor would be how many developers have consistently contributed more than 20 times over the entire project's duration. This is shown in the figure below:

Figure 4: Consistent contributors over time

It can be seen that only 12 contributors meet these criteria. This could be taken as the truck factor.

10.6.0.2 Documentation Debt

HA has a short release cycle with a major release every two weeks. Maintaining an up to date documentation in such cases can be a challenge. Observing the status of the documentation can give an indication of TD

levels. The HA's documentation is maintained in a separate repository. To ensure up to date documentation, new features and fixes are not merged without a supporting documentation update. This results in a non-existent documentation debt.

10.6.0.3   Issue Resolution and Pile Up

The speed with which issues are handled (Issue Resolution) and the number of issues which have persisted over releases (Issue Pile Up) are good TD indicators. To analyse these indicators, the following were used:

- How fast are issues closed
- What are the oldest issues still open and why have they not been closed.

10.6.0.3.1   Issue Resolution   The last 2000 closed issues were analyzed and grouped according to the time between when they were opened and closed. A histogram was created from this data and is shown below:
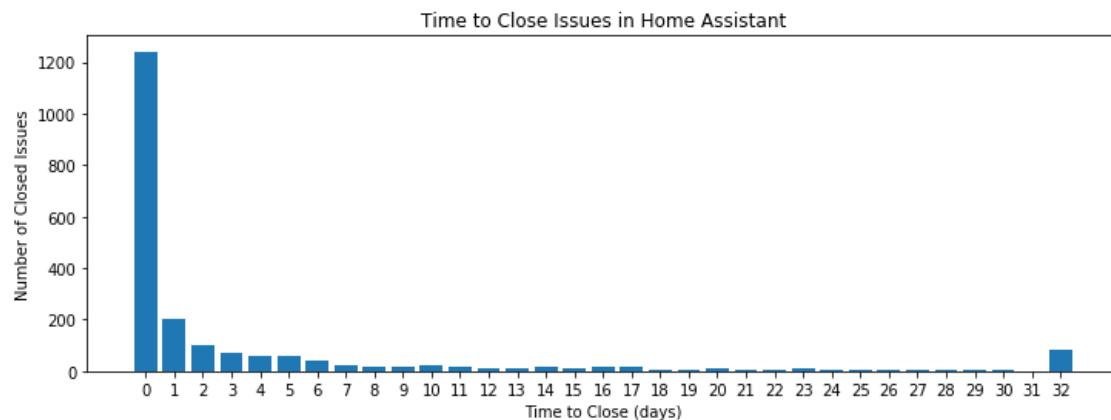
Figure 5: Issue resolution speed

The first 32 (0-31) bins indicate the number of days to close an issue while the 32nd bin indicates issues closed after more than 31 days.

More than 50% of all issues were closed within a day. This indicates timely issue resolution and a low level of TD.

10.6.0.3.2   Issue Pile Up   An issue was counted as having piled up it has been open for more than two years. The table below gives an overview of open issues older than two years and also includes the number of closed issues for reference:

Year	Issues Open	Issues Closed
2014	0	8
2015	0	282
2016	3	1428

Year	Issues Open	Issues Closed
2017	33	2462

Table 3: Open/Closed issues per year

From the table above, it is seen that the number of open issues are less than 10% of the total number of issues per year. This generally indicates a low level of TD. However, the analysis for 2018 and 2019 (so far) shows that there is an increase in the number of issues left open as seen below:

Year	Issues Open	Issues Closed
2018	440	3160
2019	486	665

Table 4: Open/Closed issues most recent years

An overview of these issues shows that many of them relate to external integrations and not the HA core. They do not adversely affect the running of the system as a whole.

10.6.1 Testing debt

Observing how much of a projects code is covered by existing tests and what kind of testing methods is a good measure of TD.

10.6.1.1 Testing practices

Testing in HA is done using `tox`. As part of the requirements for new features and fixes, all existing tests must pass. Also, see Standardization of Testing.

10.6.1.2 Test duration

The test suite is also run for different versions of Python. As a result, a full test suite run takes more than an hour. The development guidelines however advise that tests can be run for a single Python version to reduce the running time. This is acceptable because the continuous integration pipeline runs the full test suite before a pull request is reviewed.

10.6.1.3 Code Coverage

Code coverage is a measure that indicates what percentage of the code base is covered by tests. The code coverage for the main (HA) project has a 94% coverage for over 1000 files as can be seen below:

Figure 6: A screenshot from the Coveralls.io HA page

To put this in perspective the highest rated Python code coverage project on coveralls.io is 99%. However, this project only consist of 8 files. The next best project shows a coverage of 94% but this project consist of only 47 files.

10.6.2   Home Assistant TD in Metrics

In addition to evaluating HA on the above mentioned factors, the repository was also analyzed using some TD tools. This section presents the results from these tools.

10.6.2.1   Sonarqube

The HA project was analyzed using Sonarqube and the results are shown below:
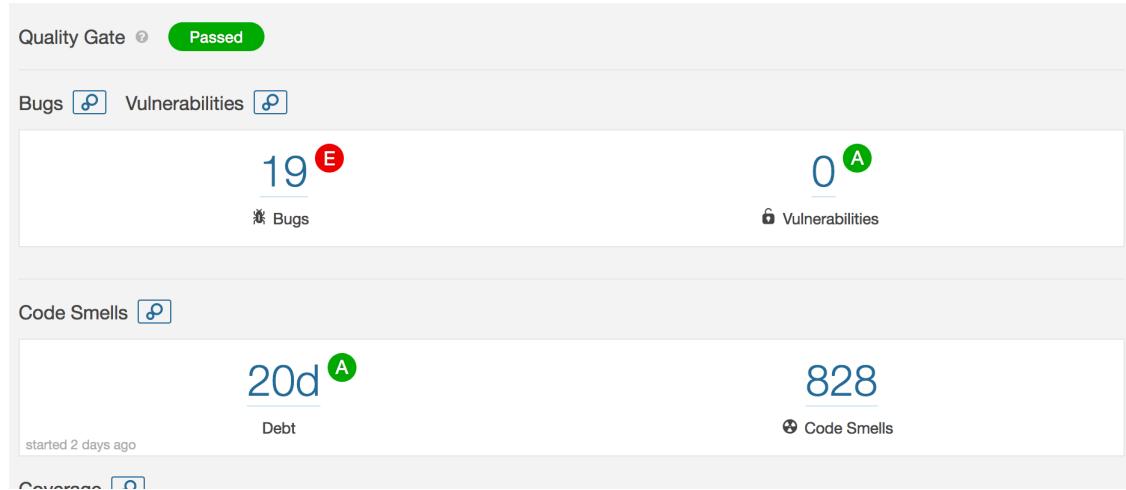
Figure 7: A screenshot from the Sonarqube HA page.

In summary, HA received a passing score for bugs, vulnerabilities and code smells.

10.6.2.2 Codebeat

HA was also analyzed using Codebeat which is an automated code review application. After the analysis, HA was assigned a 3.63 GPA (on a 4.0 scale). In terms of code complexity, only 2% of the files were deemed as *complex*. A summary of the analysis is shown in the image below:
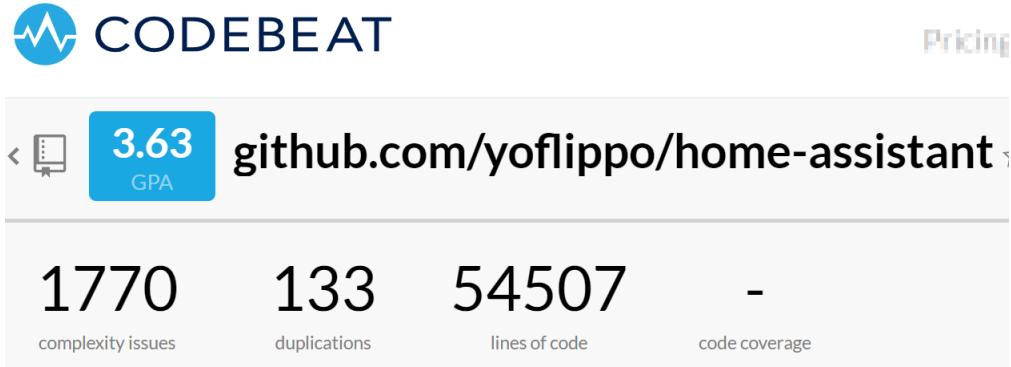
Figure 8: A screenshot from the Codebeat HA page. This is based on a fork of the HA project.

10.6.3 Mitigating TD

The core developers of HA have taken measures to ensure that TD is kept at a minimum. Some of these measures are stated here:

10.6.3.1 Code Style

The developer's documentation contains a section that states which style guidelines are to be adhered to. Also, as part of the code review process, Hound, an automated code review application, is used to check style violations.

10.6.3.2 Code Reviews

Every new feature or fix, in addition to conforming to the development checklist and passing all tests, is also reviewed extensively by contributors to the project. Merging happens only after approval has been given.

10.6.3.3 Code Owners

For a large project like HA it is difficult for a single person to have an in depth view of the internals of every component. This introduces difficulty with code reviews, i.e. who is the best reviewer for a feature/fix. To solve this problem, HA uses the Code Owners concept which indicates contributors responsible for each part of the project. Code owners are automatically notified when changes are proposed to their parts. This ensures that the contributor with the best knowledge reviews a feature and reduces the possibility of introducing further debt.

10.6.3.4   TD Awareness and Accepting Feedback

Despite the checks in place, core team members also realize that the landscape of possible TD sources increases with HA's growth. There is also a general readiness to accept and propose constructive feedback on the "right" way to implement a feature to avoid debt introduction.

10.6.3.5   Evolution of TD

To properly evaluate the evolution of TD, an empirical analysis was performed. 8 past major releases were analyzed using Sonarqube. The results are presented in figure 10.

These analyzed versions span from 2016 to 2019. From the data it can be seen that while the overall TD, as measured by Sonarqube in days, has increased steadily this increase has been very low. Also, the number of identified bugs in the current version is lower than that of the earliest analyzed version. It is logical that the debt associated with the project would grow as the project grows in size but from the foregoing analysis it can be concluded that the development practices employed in the project have kept this debt growth to a bare minimum.

10.6.3.6   Historical Analysis of TD and Corresponding Impact

To obtain a historical analysis of the TD, an empirical study was performed. 8 past major releases spanning from 2016 to 2019 were analyzed using Sonarqube. The results are shown in figure 9 below:

Version	LoC (k)	Bugs	Smells	LoC (k) / Smells	TD (days)	LoC (k) / TD	Duplications (%)
0.3	75	35	307	24.4	6	12.5	3.3
0.4	119	117	438	27.2	10	11.9	1.7
0.5	148	134	508	29.1	12	12.3	2
0.6	185	151	587	31.5	14	13.2	2.5
0.7	204	102	664	30.7	16	12.8	2.5
0.8	245	22	792	30.9	17	14.4	2.2
Current	304	19	828	36.7	19	16.0	2.3

Figure 9: A table with SonarQube metrics (and derivations) with regard to different project versions.

10.6.4   TD Conclusion

From the data it can be seen that the overall TD has been relatively low has been improving. Considering the size of the project and the number of developers this is exemplary.

10.7   Development view

This view of the HA project describes how the software is composed from a development perspective. A condensed overview is given of the most important models, procedures and standards that enable and support

the development of HA in an efficient and structured manner.

10.7.1 Module Organization

The HA source code is organized in to several modules that encapsulate specific functionality. Below, we present HA's module organization diagram with their dependencies.
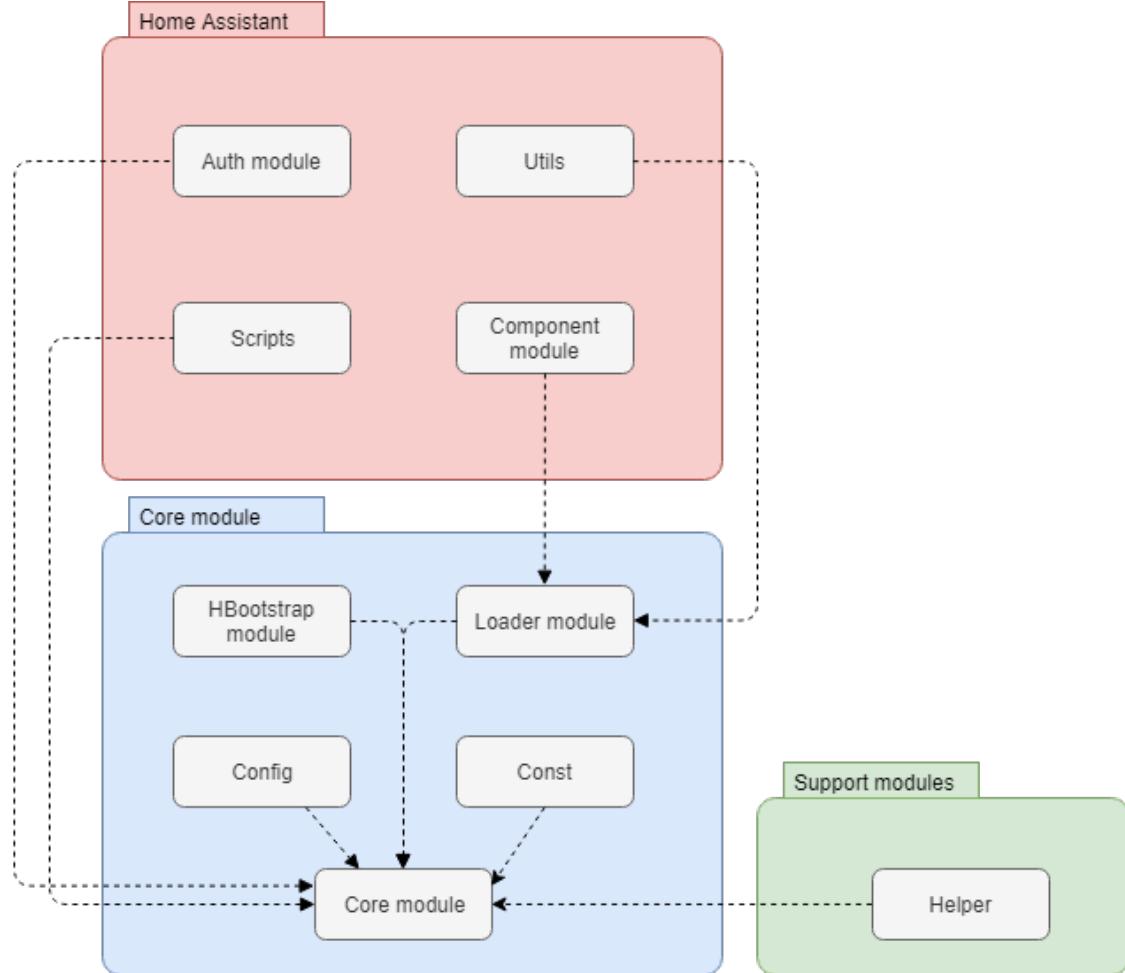
Figure 10: Module organization.
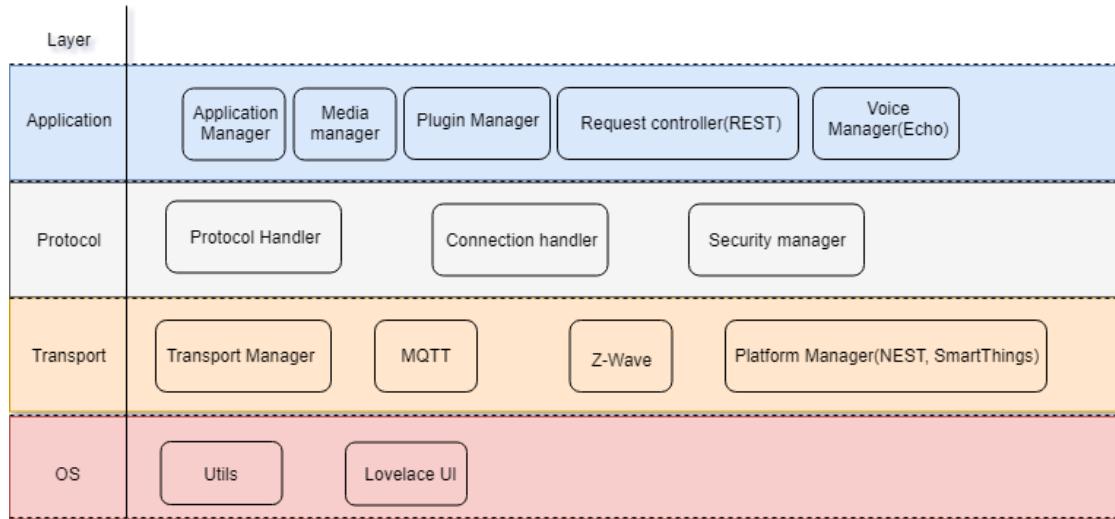
10.7.1.1 Home Assistant modules

Module	Description
Util module	This module contains common utility functions that can be used by other platforms and components throughout HA. These include: finding the current location, calculating the distance, getting the temperature and more.
Helper module	The helper module is a collection of unrelated methods and is always instantiated as a singleton. In this HA project, the helper module contains functionality for validation, discovery and other helper functions for various components.
Bootstrap module	The bootstrap module has the following functionalities: Set up the logging, try to configure HA from a configuration dictionary stored on disk and start all components. The setup dynamically loads required components and its dependencies. It also tries to start all enabled components that are found in the configuration.
Config module	The config module checks if the HA configuration file is valid and parses the configuration if it is. This configuration contains all the information of which components and settings are enabled in HA. When there is no configuration file found a default configuration will be used.
Const module	The const module is using to constants used by HA components.
Core module	The core module is the main module of the HA system, which uses synchronous and asynchronous methods to deal with all inputs and outputs.
Exceptions module	The exception module contains all custom Exceptions that are used throughout HA.
Loader module	The loader module is using for loading HA components. A quote from the developers:Components can be accessed via `hass.components.switch` from your code. If you want to retrieve a platform that is part of a component, you should call get_component (hass, `switch.your_platform`). In both cases the config directory is checked to see if it contains a user provided version. If not available it will check the built-in components and platforms.

Table 5: Home Assistant modules

10.7.2 Common Processing

Common processing approaches address aspects of the architecture of HA that require a standard approach across the whole system. This is a key task as it contributes to the overall technical coherence of the system and clarifies how and where processing is done.

Figure 11: Home Assistant layers overview.

10.7.2.1 Modularity

The HA project contains multiple directories like home-assistant.io, docs for documentation, tests for the homeKit accessory-information service tests, Github for the PR template, scripts for project scripts, and virtualization for environment setting.

10.7.2.2 Application layer

The application layer of HA is used to separate the web layer from the application layer (also known as the platform layer). This allows you to scale and configure both layers independently.

10.7.2.3 DB layer

The database layer in HA allows you to write and read data between other layers and database. The default database used by HA is SQLite, and the database file is stored in your configuration directory the DB layer (SQLite) is the common processing for the HA server.

10.7.3   Standardization of Design

10.7.3.1   Internationalization

The HA internationalization project includes preparing platforms and the frontend for localization, as well as the actual translation of localized strings. Platform translation strings are stored as JSON in the home-assistant repository. Custom components cannot take advantage of Lokalise for user submitted translations.

10.7.3.2   Maintainability

The documentation of HA are very important to record each product demand and issues. To standardize the software design, the documentation aspect should be considered, in HA, the language, word formatting, text style also should be objective and not gender favoring, polarizing, race related or religion inconsiderate.

10.7.3.3   Renaming Pages

Renaming is also standardized. It can happen that a component or platform is renamed, in this case the documentation needs to be updated as well. If you rename a page, add redirect_from: to the file header and let it point to the old location/name of the page.

10.7.3.4   Template

The HA project makes the template for formatting documentation. All examples containing Jinja2 templates should be wrapped outside of the code markdown, and some static file are not allowed be used in template, such as `states.switch.source.state`.

10.7.3.5   Technical cohesion of the system

HA uses Python as development language, and implements commonly used middleware, external API's (REST API, WebSocket API, Server-sent event) for extension.

10.7.4   Standardization of Testing

The HA project uses **tox** to standardize and automate their tests. This wrapper sets up the virtual environment and runs all tests, including validation of code style and documentation style. The tests are written using the Python unittest module.

The project uses **Travis CI** as continuous integration tool. This is set up to run on every pull request and tests against different versions of Python 3 to ensure that the release will be stable across different versions. When the test suite has completed all tests, the code coverage is checked by **Coveralls**.

Also, the guidelines and testing specifications stipulate that new features must include tests and where not possible (as in the case of features depending on external devices) should be explicitly excluded from being tested.

10.7.5 Instrumentation

The most important monitoring tools available are the logging system and the server-sent events stream. For the logging HA makes use of the very popular Python module **logging**. To tune the verbosity of logging you can change the log-level, because most of the time verbose logs are unnecessary.

The **server-sent events stream** is available to subscribe to as a client. This provides a way to receive messages without Python logging to monitor HA.

10.7.6 Codeline organization

The codebase of the HA can be found on the Github platform and is fully opensource. To develop features, fix issues or contribute code you will have to set up a **development environment**. In this development environment one should use a fork of the upstream **dev** branch as base. By creating a Pull Request to the home-assistant dev branch on github Travis CI is triggered the pull request to run the all the tests in the suite. Next to this, automated code review tool HoundCI is triggered by a pull request as well to check for code-style violations.

The HA project makes a new release every 14 days on average. These releases include the latest bug-fixes, features and components that have been included and approved by founder @balloob. Before anything can be included in a release it has to pass all tests in the test suite and pass the tests of pydocstyle, pylint and flake8. These tools are used to comply with the PEP8 standard and keep the code and docstyle compliant with the PEP8 and PEP257 standards. The releases are built using custom tools which provides the different options to install HA. The most important builds are: HassOS Docker Hypervisor, HassIO the private cloud supervisor and the HA program itself. HassOS and HassIO are not required to run the HA program but make it easier for user to install and deploy HA and add-ons. HA is not built as it is a Python program.

Subject	Description
Source File Organization	Divided into platforms, components and a core. Components and platforms can be enabled to run using the configuration.
Module groups	Every component is organized to be a Python module which can be imported.
Directory structure	The top-level directory consists of *docs, homeassistant, tests, script* and *virtualization*. The homeassistant directory contains the core and the directories for the platforms/components.
Building the source	The homeassistant program itself is not built, as this is a Python program there are no binaries that are built and released. Instead, the code is released and run using a Python 3 interpreter.

Subject	Description
Scope, type & running of tests	Almost every part of HA is unit tested. Next to the unit testing the code and documentation is validated using tools which are configured to all run using *tox*.
Source release	The HA program is released on github and continuously tested using Travis CI. Other parts of the HA project such as HassOS are built and then published on the website.
Source control	Source control is done using Github. The HA project is organized into multiple repositories, each with a different purpose, such as for example the backend or the website. Development is done by creating pull requests against the upstream *dev* from your own forked repository.

Table 6: Codeline Model Overview

10.8   Functional View

This section describes the functionalities, capabilities, interfaces and interactions of the HA project.

10.8.1   Functional Capabilities

From a high-level overview one could say that HA has three important capabilities, these are:

1. Receive and process events, which includes messages from sensors connected to HA.
2. Control devices and send events to other devices such as notifications or status updates.
3. Provide a centralized interface and an overview of everything connected to HA with a web-accessible user interface.
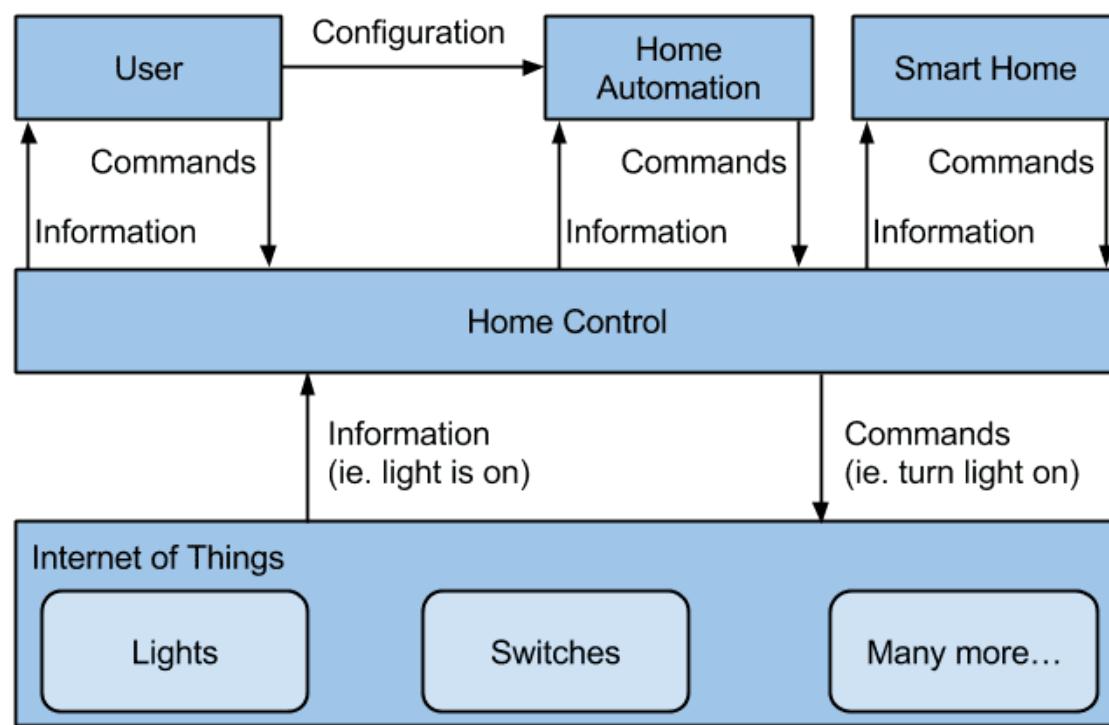
We can see how these functionalities of HA interact in figure 12 below. The Home Control part of the architecture is the core of the architecture. The user can interact with the Home Control using one of the interfaces provided by The HA project, more on this can be found below in the interfaces section.

Figure 12: Home Assistant high-level functionalities.

10.8.1.1   Core Functional Components

As seen in the figure above, Home Control is central to HAs functionality and is referred to as the **core** of HA. This core functionality is split into four major functional categories:

- Event Bus
- State Machine
- Service Registry
- Timer

Figure 10.3: The HA landscape

Event Bus HA relies heavily on an event based architecture. The core part of HA provides an event bus which facilitates communication between components, the core and other external entities. Events are triggered i.e placed on the bus by connected devices/services for instance when some action is taken by the device. Similarly, other connected devices can listen in on the bus for events from interesting devices and take action based on such events.

State Machine Central to HA being able to control connected devices and provide an overview of the state of the connected devices is its ability to manage the device state. The State Machine provides a store for the state of each connected device. When a device state is changed, it triggers an update of the state machine. The state machine updates its internal store and fires an event on the bus with the updated state.

Service Registry Connected devices might offer services. As an example, a connected media player would offer a play/pause/stop service for media files. The service registry allows such devices to register the services that they offer as well has handlers for such services. Users of the system or other connected devices can also request for registered services using the interface provided by the Service Registry.

Timer The Timer serves a function analogous to a clock in a synchronous processor. It fires a `time_changed` event every second. Connected devices can listen for this event and use it to schedule state polling or other recurring actions.

10.8.2   Design Philosophy

According to the vision of founder Paulus Schouten (@balloob) HA should run at home and everything you automate should work flawlessly. Therefore, the design of HA is as modular as possible, allowing to keep the installation as small as necessary while keeping the system maintainable and extendable. The architecture and technologies used in HA follow this vision, which results in HA scoring high on design qualities such as separation of concerns, cohesion and coupling.

10.8.3   Home Assistant Extensions

While the Home Control part of the HA interface is responsible for controlling/managing connected devices the HA Extensions provide the interfaces with which devices can be controlled and managed. These extensions are called **Components** in the HA ecosystem. There are two broad categories of components:

- IoT Domain Components
- Task Domain Components

IoT Domain Components These components interact with connected devices. They update device state in the state machine and expose services offered by the devices via the service registry.

Task Domain Components These components are more automation centric. They typically listen for events and state updates from IoT domain components and based on pre-defined logic perform some action. An example might be a `play_wakeup_music` component which triggers the play service of a connected media player on sunrise.

10.8.4 Interfaces

HA provides a number of external interfaces for interaction with the system.

10.8.4.1 Lovelace

Lovelace is a web based graphical user interface with which users can manage connected devices. It allows the user to add new devices, request for registered services, monitor device state, etc.

10.8.4.2 Hass.io Command Line Interface
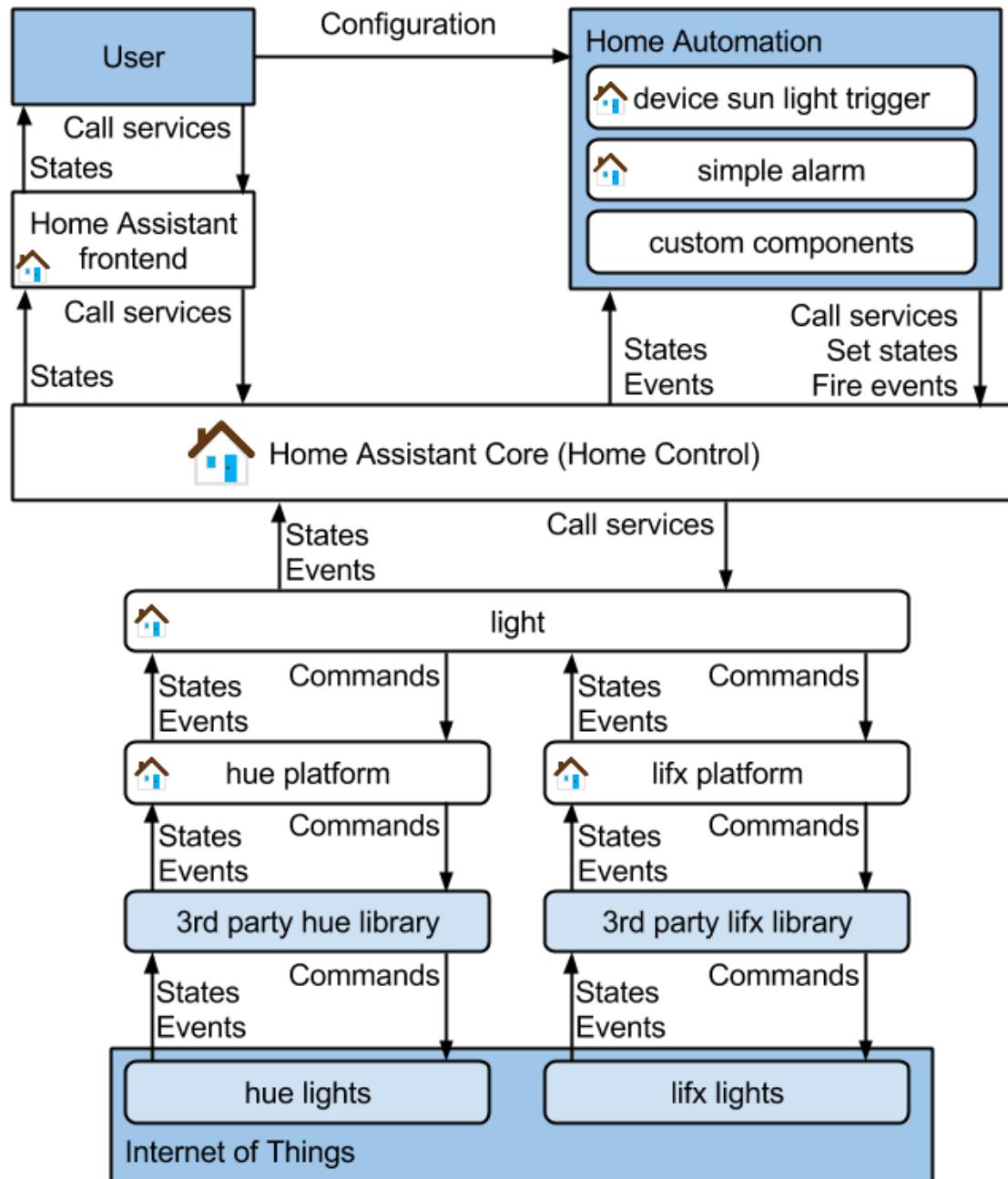
HA can be setup on any operating system but Hass.io, which was designed with embedded devices like the Raspberry PI in mind, simplifies the setup process. It provides a command line interface which allows the user to start or stop HA, view generated logs or query the state of connected hardware.

10.8.4.3 IOS Companion App

HA supports integration with a companion IOS application. This interface exposes information about connected devices and allows the user to manage devices and update configuration entries.

10.8.5 The bigger picture

When combined, the capabilities, core components, extensions and interfaces together give an overview of capabilities and functionality the HA system as a whole. This can be seen in the image below.

Figure 13: Home Assistant full architecture schematic.

The core is made accessible by the interface options while the separation of the core and components make it easier to extend the core functionality. Next to this, a user can configure the system to trigger on events

that are sent over the event bus and display information in the frontend of HA.

10.9 Conclusion

This chapter analyzed a number of aspects of the Home Assistant project. It can be concluded that the architecture is well-designed and HA is growing in popularity and number of active and contributing users.

The different stakeholders in the Home Assistant project were extracted and presented. Next, the Context View revealed the interactions of the project with different entities. The TD, Development View and Functional view then shed light on how the system is developed, maintained, tested and used.

In Conclusion, despite the rapid growth in terms of size and number of components the system remains clearly structured and is well maintained. Next to this, the guidelines, documentation and active community make it easy to contribute to the project which in turn supports the project's growth.

10.10 References

- [1] Rozanski, N., & Woods, E. (2012). Software systems architecture: working with stakeholders using viewpoints and perspectives. Addison-Wesley.

- [2] Apache License, Version 2.0, January 2004. http://www.apache.org/licenses/LICENSE-2.0.txt

- [3] Gardner, J. R., Rachlin, R., & Sweeny, A. (1986). Handbook of strategic planning. John Wiley & Sons.

- [4] https://codescene.io/projects/4449/jobs/12494/results/social/authors

- [5] https://www.linkedin.com/in/schoutsen

10.11 Appendix

We analyzed 10 Pull Requests (PR) to identify stakeholders and decision-making process. The PRs were selected by looking at the most commented closed PRs on github and selecting 5 rejected and 5 merged ones. The PRs that were used in this analysis are listed below.

Approved PRs

Pull Request	Description
Rflink gateway addition	PR was opened by @aequitas with the code still in a rough state as WIP. A bot automatically assigns reviewers, one of which was @balloob (Founder of Home-assistant). Code owners are marked in the a separate file and when working on components of a code owner he/she will automatically be triggered to review the PR. @balloob provided extensive reviews and requested some changes. All of these changes were addressed over the course of two months. The size of the pull request resulted in some inefficiency because comments were missed and addressed late because of it. Comments reveal that there is too much documentation that contributors do not read or are unaware of, because patterns are introduced that they do not want (They do not want communication over event bus)
Added new climate component from Daikin	Interesting dynamic in this pull request. This time @tinloaf provided an extensive review and requested changes, next to that there were quite some lint errors. It turned out the user was new to Python and Home Assistant and got a little impatient. For instance, the device that was added appeared to have multiple interfaces. And the person who opened the PR (@rofranz) had opened another PR. The community however, told @rofranz to first finish the current PR. Because of the size of the commits @tinloaf wanted someone else to do a review as well. Which resulted in a whole lot more suggestions and requested changes. These were all addressed by the user and the PR got merged. So quite a successful review structure and collaboration as well as a good experience for the user in the end.
Improve InfluxDB	Pull Request reverting and fixing issues. Interesting discussion on data types and improvements for the DB. Migration script ended up in a breaking change which in turn got fixed.

Pull Request	Description
Add deCONZ component	deCONZ is software that communicates with Raspbee and Zigbee gateways and exposes the connected devices. The user @Kane610 opened this PR to add functionality for the HA. Some discussion between @balloob, @Kane610 and @tinloaf took place about naming conventions and on how I/O should be addressed (never inside the events loop). Later a big discussion was started about implementing it 'entities' or 'events'. E.g. should a button be an entity or should it be implemented in an event (the latter was chosen). Finally @MartinHjelmare merged the commit.
Xiaomi vacuum as platform of new `vacuum` component derived from ToggleEntity, and services	In this PR a vacuum component is added. The discussion mainly revolves around the way a component has to be implemented. During development of the PR @MartinHjelmare had some real critical pointer to the creator of the PR @azogue. At the end, several members of the community applauded the work of @azogue. Finally @pvizeli merged this PR but the result was a flaky test as spotted by @balloob.

Not-Approved PRs

PR	Description
Add new platform AndroidTV/Android	This pull request attracted some attention because a number of people wanted the feature. It looked promising and almost ready to merge, but then got stuck on python-adb dependencies. Home Assistant founder @balloob requested changes to fix the dependencies and create a new PyPi package and closed the pull request. He suggests opening a new PR once the package has been fixed meaning this PR is closed and not merged.

PR	Description
Add Component for PlayStation 4 consoles as integration/config flow	New component for the Home Assistant. Reviews resulted in a lot of requested changes which all got addressed. At first the code was written to support multiple PS4 devices. However, this resulted in a few issues and to get the component to be included this code was omitted as they decided that not many people have multiple PS4 devices. So they omitted that code and merged the working code for one device, leaving the multiple device implementation open for now. Most important review work was done by @MartinHjelmare, one the active collaborators for Home Assistant.
Improve Philips Hue color conversion	The original problem description is not clear. In a discussion between @starkillerOG and @amelchio the problem is clarified as that the PR tries to fix is the limited amount of colors that can be chosen from within the HA. Finally the PR was closed as the solution was publish in another project (aiohue library).
Cover component for RFlink	This PR was an attempt to create a new HA component for the Somfy RTS more specifically the RFlink. The PR was started by @passie which has not opened a PR in the past. Some problems are encountered with the signing of the CLA. Then the PR is dominated by @houndcli-bot (a bot) which reports a lot of unused statements in the code of @passie. Some version of the code is tested by @olskar which is reported to work. The PR closes with another bot statement reporting that a commit was placed via a non-linked github account. Finally, @passie closes the PR.
Add support for some Tuya devices.	The Tuya is an IoT *platform* to wirelessly connect home devices. The person who opened the PR (@sean6541) want to design a *component* for the HA. Actually, some further research shows that the Tuya is implemented in the HA, contrary to the final state of this PR. The houndci-bot initially had a lot of remarks. After some discussion and new commits @sean6541 is showing up less frequently and is not seen anymore. @pvizeli closed the PR for reopening when all open comments are addressed.

PR	Description
Rfxtrx binary sensors	This PR is an attempt to add a Rfxtrx sensor ass a component to HA. This is a transceiver for different devices in the area of domotics. The PR started with a lot of linting comments from @balloobbot. Some discussion about the implementation and testing happened. Then @balloob closed this PR because it had gone stale by @ypollart. A few months later @ypollart reopend another PR to add this sensor.

Chapter 11

IPFS

Figure 11.1: GO-ipfs

11.1   Contents

11.2   Introduction

The Internet and The World Wide Web are considered to be some of the greatest inventions of the 20th century. However, while the Internet is built on the principles of decentralization from the ground up, we cannot say so about the Web itself. The majority of the websites is operated by a few hosting companies and

most of the static content is served by a few *Content Delivery Networks (CDNs)*. For example, Cloudflare[1] currently powers 10% of all internet requests. Since there is a central point of failure, if a provider experiences outage, a large part of the internet goes down. Another issue with the current infrastructure is a *crisis of erased history*. One cannot expect that a web address visited today will still work in several years from now or that the content will stay the same. In fact, the average lifespan of a webpage is only 75 days[2]. Last but not least, the current web is not censorship-resistant, and some governments already intentionally restrict their citizens from having access to information[3].

These were probably the main motivations why **Protocol Labs** was founded in May 2014 with the mission to improve Internet technology. They started as a YCombinator company and released an initial version of the **InterPlanetary File System** (IPFS) in January 2015. IPFS combines the best ideas of Git, BitTorrent and Kademlia to provide a truly decentralized, peer-to-peer hypermedia protocol. In 2016, several modules evolved into separate projects, namely **libp2p**, **IPLD**, and **Multiformats**, all of which constitute the building blocks of IPFS.

The *InterPlanetary* part of the name pays a homage to Licklider's Intergalactic Computer Network[4], a computer network concept which inspired the development of today's Internet. In theory, IPFS could one day be actually used on an interplanetary scale, thanks to its peer-to-peer architecture and ability to work in disconnected networks.

11.3   Structure of IPFS Project

The IPFS project is well structured into teams to separate responsibilities. There are three project roles that are responsible for the long term vision and coordination of teams – Benevolent Dictator for Life (BDFL), IPFS Project Lead, and IFPS Project Coordinator. These roles are further explained in the stakeholders section. The teams of the project are split into two types: working groups (WG) and research groups. The working groups are developing and deploying software, while the research groups are exploring new areas. Some former working groups (libp2p, IPLD, and Multiformats) have grown to become their own fully separated projects.

The whole IPFS project consists of more than 200 repositories, but in our architectural description we focus on go-ipfs, which serves as a reference implementation of the IPFS protocol. Go-ipfs being a working group of the IPFS project has multiple relations with groups and project roles that are not part of the go-ipfs working group. This is further explained in the power grid.

11.4   Stakeholders

In this chapter, we will identify stakeholders involved in the go-ipfs project using the categories according to Rozanski & Woods [5], specify managing roles in the overall IPFS project and its relation to the go-ipfs project.

---

[1] https://www.cloudflare.com/learning/what-is-cloudflare/

[2] http://clgiles.ist.psu.edu/papers/Computer-2001-web-references.pdf#page=5

[3] https://blog.ipfs.io/24-uncensorable-wikipedia/

[4] https://discuss.ipfs.io/t/why-the-name-ipfs/307

[5] Nick Rozanski and Eoin Woods. Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives, 2nd Edition. 2011.

IPFS Project

Project roles

BDFL Project Lead Project Coordinator

Teams

Working Groups

Project Management
Community
JavaScript Core Dev
Golang Core Dev
IPFS GUI
IPFS Cluster
IPFS Infrastructure
Integration with Web Browsers
Dynamic Data and Capabilities
Decentralized Data Stewardship
Local Offline Collaboration
Package Managers

Research Groups

CRDT
Blockchain Data
Bitswap
p2p-video
pubsub
dht

Related separate projects

libp2p

IPLD

Multiformats

Figure 11.2: IPFS Structure

11.4.1 Roles

To properly define roles, we inspected the Protocol Labs profile on Crunchbase and various IPFS repositories on GitHub including ipfs-mgmt, where we learned how the IPFS community operates and how it is structured. To identify people most involved in the process of creating code in the go-ipfs project, we analyzed 20 recent (created after 1 September 2018) merged and unmerged Pull Requests with the highest number of comments.

Pull Requests we used can be observed in the following table:

Type	List of Pull Requests
Merged	#5526, #5611, #5789, #5472, #5785, #5455, #5864, #5659, #5501, #5649
Unmerged	#5474, #5583, #5435, #5840, #5464, #5744, #5479,#5570, #5514, #5849

The analysis of these Pull Requests provided us with knowledge which developers are actively taking part in the process of code review and also that almost all Pull Request have to be finally accepted and merged by the Captain of go-ipfs (@Stebalien).

It was also helpful to explore issues used for team discussion like a reflection issue or logs from weekly meetings between members. From this, we noticed that the project is developed using Agile approach. The team uses quarterly Objectives and Key Results (OKR), which helps them to communicate goals and achievements on a quarterly basis. Developers work asynchronously and remotely, however, each week there is a weekly call to discuss project related issues, such as the current state of their work and plans for the future.

As contact people, it would be the most helpful to contact developers which participate most actively in Pull Requests and also contribute a lot, as they would possibly have the most significant knowledge about the system structure and various issues they often have to overcome. We would choose @Stebalien, @Kubuxu, and @magik6k.

Results obtained from these various sources can be found below.

Type	Stakeholders
Acquirers	Acquirers are sponsors and distributors of the product. The acquirer of this project is Protocol Labs.
Assessors	These stakeholders are responsible for overseeing conformance to standards and legal regulations. Assessors in the go-ipfs project is Protocol Labs legal department.
Communicators	The interest of this stakeholder is to explain the usage of the system via documentation and training. There are two recent and popular conferences: IPFS Conf 2019, MIT Bitcoin Expo 2018.
Competitors	ZeroNet, Storj, Burst, Genaro, MaidStafe, Dat

Type	Stakeholders
Developers	Most of the people involved in work on IPFS are employed by Protocol Labs. People responsible for developing new functionalities and maintaining the project are currently: @Stebalien, @magik6k, @Kubuxu, @hsanjuan, @kevina, @hannahhoward, @eingenito.
Investors	Investors are the stakeholders providing resources to make it possible to continue the development of the system. The main investor of the project is Protocol Labs (mostly sponsored by Union Square Ventures, BlueYard and YCombinator), other investors of the IPFS project are FundersClub, Digital Currency Group.
Maintainers	These stakeholders are responsible for the evolution of the project. The software is in alpha version, however, people trying to maintain the long-distance vision are @daviddias, @Stebalien.
Suppliers	Build or supply hardware and software on which system is working. In this project software used is mostly written in Go and Bash.
Support staff	IPFS has a designated community project for educating about how the IPFS and its tools work and can be exploited. The Captain of the community project is @mikeal.
System administrators	System administrators are people running the software once it has been deployed, and as an idea of IPFS is to be a P2P network, users are the most obvious stakeholders of this type. On the other hand, IPFS project has a designated team of engineers (infra-team) in charge of deploying and maintaining core system infrastructure.
Testers	These stakeholders test if the system is working and ready to be deployed. These are developers and users using a designated TestBed.
Users	This group of stakeholders uses the product and has concerns about its functionality. Representants of this group include: web developers who want to host static files, dapp developers, users who want to access data stored on IPFS, pinning services (Eternum, Pinata), gateway providers (Cloudflare), and hosting providers (Neocities).

11.4.2 IPFS Roles

IPFS has some designated managing roles which are explained briefly in the table below. In the scope of this chapter we mainly focus on the Go implementation, however, to properly understand how the management processes works in the go-ipfs project it crucial to observe everything in the bigger picture, namely the whole IPFS project and all of its subprojects. It is interesting to see how these projects collaborate with each other to produce the final product.

Type	Stakeholder	Description
IPFS BDFL	@jbenet (the original creator of the IPFS Project)	**Benevolent Dictator for Life**: Lead the IPFS Project at a long term scale. Represent the IPFS project to a multitude of communities. Take responsibility for setting the direction of the project. Set the key priorities for the project.

Type	Stakeholder	Description
IPFS Project Lead	@daviddias	**Captain of the Captains**. Align teams to build solutions for technical challenges. The lead director of the implementation of the protocol.
IPFS Project Coordinator	@momack2	Ensure that teams rely on a uniform structure with respective customizations as needed. Create and maintain the platform for resource allocation across teams and projects. Ensure that there is a steady communication flow between teams and individual contributors.
Golang IPFS Captain	@Stebalien	The Captain takes the lead on writing or guiding the conversation specs, documentation, and other artifacts to support the team. The Captain is also the gatekeeper of the Working Group Roadmap and accumulator of the Working Group Knowledge, guiding the group to make good decisions.
Golang IPFS Technical Project Manager (TPM)	@eingenito	The Technical Project Manager (sometimes referenced as Project Manager, Program Manager, and cat herder) is a team enabler. TPMs own the Quarterly Planning process including OKRs and Retrospectives. They ensure that the coordination strategy the WG selected is well executed.

From the table above we can observe that important decisions related to IPFS are supervised by the **Benevolent Dictator for Life**, the IPFS Project Lead and all Captains of all projects. The Project Captain and Technical Project Manager are enforcing and supervising the implementation of new features or changes in their project. Developers of go-ipfs propose changes using Pull Requests, but each of them has to be accepted by one of the project supervisors (@Stebalien - Captain or @eingenito - TPM). Or when changes are more significant, higher instances are involved (@daviddias - captain of captains or even @jbenet - BDFL).

11.4.3 Power-Interest Grid

To visualize the power and interest of stakeholders, we created a power-interest grid. Stakeholders with the most power and interest are located in the top-right quadrant. The Golang IPFS Captain and TPM are in charge of the direct course of the project. The developers are also located in this quadrant, but have less power.
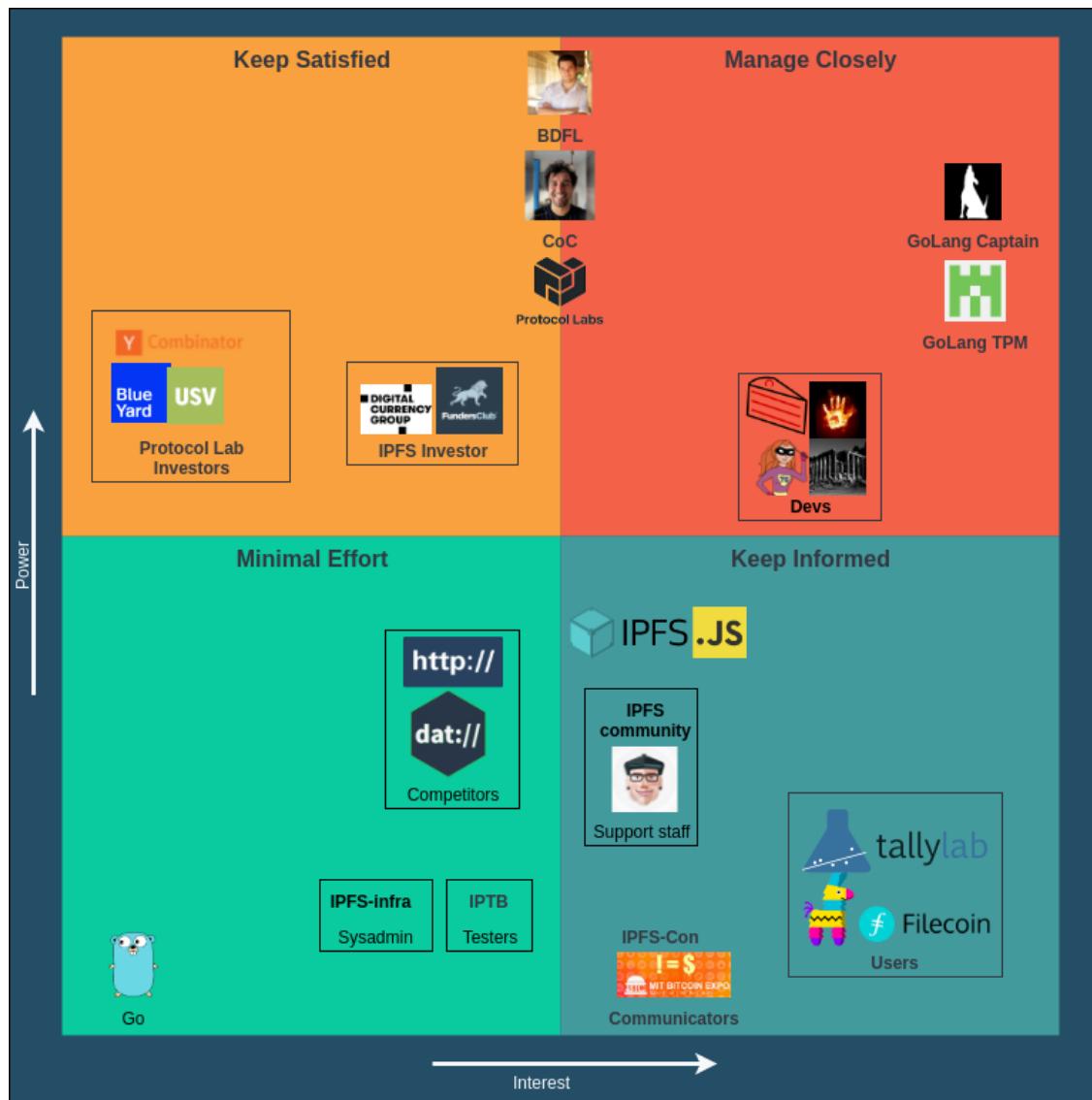
Figure 11.3: Stakeholders Power Grid

In the top-left quadrant, the investors can be found. The BDFL and the Captain of Captains are somewhere in between the top left and right quadrants. They are in charge of the overall vision and progress of the general IPFS project. As they make top-level decisions, they have a lot of operational power, but are less involved in the specific implementation of the go-ipfs project.

The users and projects that depend on the go-ipfs project are in the bottom right quadrant. They need to be up to date with the project, so that any major change won't affect them.

In the bottom left quadrant, we have suppliers. The Go language has some influence on the project. If Go changes, the project will be affected.

11.5   Context View

The context view describes relationships, dependencies, and behaviors between IPFS and external entities.

11.5.1   System Scope

IPFS is a protocol for a distributed file system whose goal is to make content delivery truly decentralized. Go-ipfs serves as its original and reference implementation.

It consists of a daemon and a command line interface (CLI). The daemon contains an HTTP server through which the CLI and other apps can communicate. It is also possible to mount IPFS to /ipfs on the local filesystem, which allows arbitrary apps to access the content stored on IPFS without communicating with the IPFS node directly.

The Go implementation is used by end users and service providers. Web apps usually use its JavaScript implementation as it is possible to run it in a web browser without any software installed on a client.

11.5.2   Context Model

To visualize relationships with external entities, a context model diagram has been designed.

IPFS is developed by Protocol Labs, a research and development company. Its development team is globally distributed, just like the Internet itself. The main contributors and decision makers of the go-ipfs project have been analyzed in the previous section.

The development including most of the decision making happens on GitHub. There are several CI tools used for running tests and code analysis, including Jenkins, CircleCI and Travis.

IPFS depends on other projects developed by Protocol Labs, especially libp2p (a network stack for peer-to-peer applications), IPLD (Interplanetary Linked Data), and Multiformats.

For the content to be available, it has to be present at least on a single node in the network. As end-user nodes go offline frequently, there is still a need for servers to store a copy of data if there is a requirement for permanent availability guarantee. Seeing this as a business opportunity, several *pinning services* (e.g. Pinata) started to provide pinning of IPFS objects for a monthly fee. Another alternative would be to provide an incentive for users to store data of other users and keep the network decentralized. Filecoin, another project of Protocol Labs, is expected to introduce that incentive in the future.
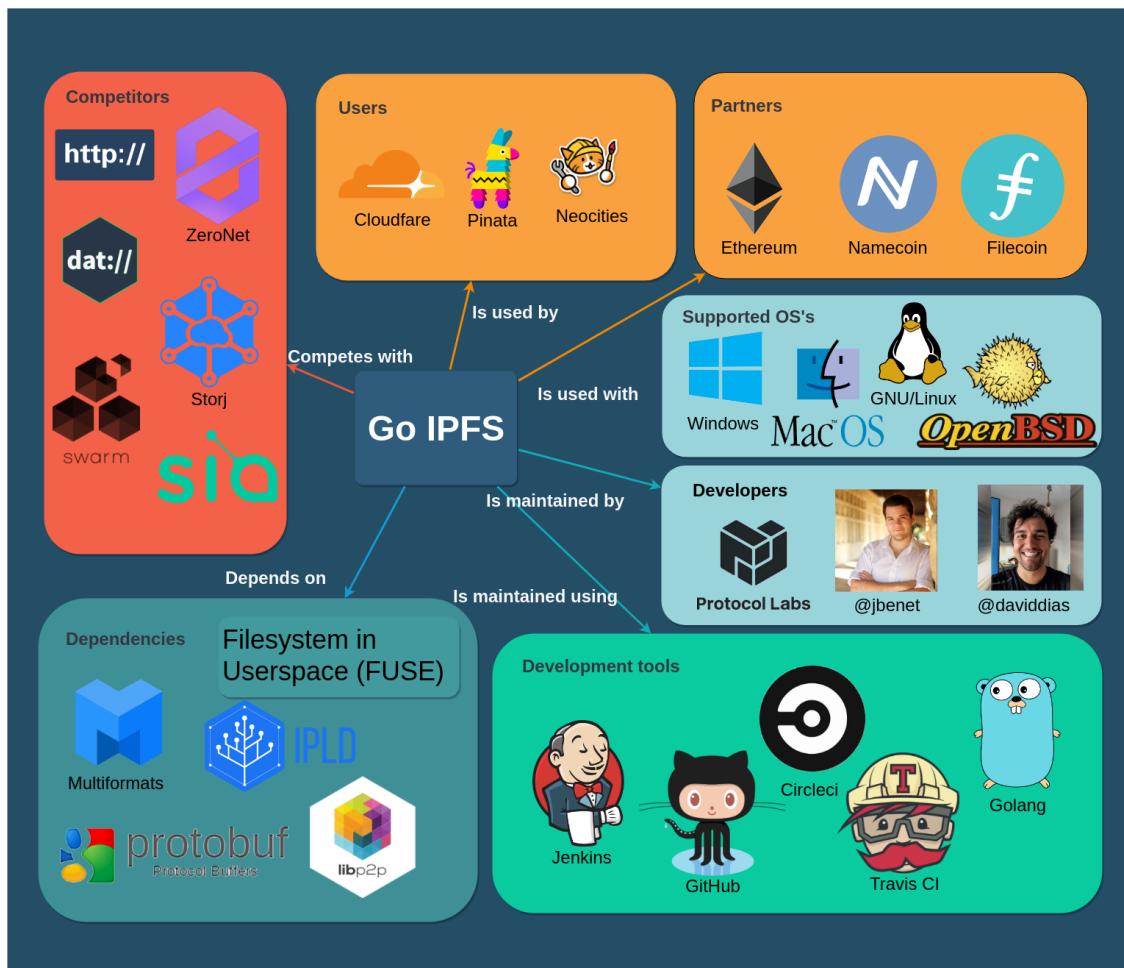
Figure 11.4: Context Model Diagram

Besides regular users, there are several parties who are interacting with IPFS. The project is interesting for website creator tools like Neocities, which are starting to use it to host static content. Cloudflare, one of the largest CDN providers, is running their own IPFS gateway and allows developers to connect an existing domain using DNS to a directory stored on IPFS. Other tools that are compatible with IPFS and aim to replace the centralized DNS are Namecoin and Ethereum Name Service.

The ultimate goal of IPFS is to replace HTTP for serving static content and make The Internet truly decentralized. For that reason, we can see the HTTP protocol as a competitor. Freenet and ZeroNet, which have similar goals, are direct competitors. Finally, any public distributed file system could be seen as an indirect competitor.

The application is multiplatform and can be run on Windows, Linux, and macOS.

11.6   Technical Debt

During our research, we found that the developers are constantly paying attention to code quality and try to maintain the quality of the software in various ways, but especially by detailed code reviews and discussions.

We analyzed the technical debt of go-ipfs in several ways. Firstly we used SonarQube, then we analyzed code manually and in the end, we decided to check if developers discuss technical debt in Pull Requests or in issues.

11.6.1   SonarQube Analysis

We use SonarQube [6] to analyze repository with regard to vulnerabilities, code smells and possible bugs.

Metric	Bugs	Vulnerabilities	Debt	Duplications	Duplication Blocks	Coverage
Value	0	0	4d 3h	1.1%	18	0.0%

As can be seen, no bugs or vulnerabilities have been discovered in the code. Also, technical debt is on a low level as having on 4 days 3 hours. Duplications are also on the low level (1.1%). Further analysis shows that the 324 code smells consist of:

- 69 TODO comments
- 12 not following the naming convention
- 47 defining a constant instead of duplicating literal
- 50 refactor function to reduce Cognitive Complexity
- 124 remove unused parameters

11.6.2   SOLID Violations

Generally it is not so easy to find violations of SOLID rules, however, function `getOrHeadHandler` in the file `go-ipfs/core/corehttp/gateway_handler.go` (226 lines) is great example of violation of Single

---

[6]https://www.sonarqube.org/

Responsibility Principle. Many parts of this large function could be extracted to separate functions to improve readability and testability.

In general case, as we noticed from analysis of Pull Requests and manual analysis of the code, developers try to follow SOLID rules and most of the function are well-written.

11.6.3 Documentation Debt

The project has two types of documentation: the folder /docs is used to explain how to use the product, while the comments in the code are used to explain the implementation and can be used to generate documentation using godoc. The code itself has very sparse documentation and some files do not have any documentation at all. After analyzing the questions on the discuss.ipfs.io website, a lot of questions concerning incomplete/missing documentation have been found. There is no visible requirement for creating documentation in the contribution guide. The project has some substantial documentation debt that can lead to a decrease in development speed and especially make onboarding of new contributors slower.

11.6.4 Testing Debt

To manage testing of such a complex repository, developers have some tools to perform that:

- unit tests stored in files with suffix _test.go
- ipfs-tb repository (IPFS own testbed)
- test module of the go-ipfs repository (this module contains online and offline tests of project functionality)

As ipfs-tb appears to be another repository which actually performs its task as testbed quite well, we decided to focus on the analysis of test module and unit tests. Detailed description of test module is done in development view.

By manual analysis of code we noticed two important things:

- many tests are not executed correctly (they should compare values instead of just logging)
- there is a small number of unit tests

Test module of the project and ipfs-tb repository are good examples of integration tests and environment for manual testing.

As SonarQube is not adding both unit tests and shareness tests in its calculation, we decided to investigate further using other tools. While investigating the CI tool codecov used by the go-ipfs team, we observed that code coverage is on the level of 62.32%.

Some modules are tested more in depth, however, the most important module of the project, core, has only 63% of code coverage (codecov is calculating ratio between lines hit and all lines). It also is the largest module of the project, however, by more detailed analysis of this module we observed that overall most classes have code coverage in the range of 60-70%. We think that this number could be improved to decrease the technical debt of the project.

Files	Number of lines	Lines hit	Lines partialy hit	Lines missed	Coverage	
📁 assets	288	113	19	156		39.23%
📁 cmd	869	518	71	280		59.60%
📁 commands	99	72	7	20		72.72%
📁 core	8,035	5,084	924	2,027		63.27%
📁 dagutils	263	157	22	84		59.69%
📁 exchange/reprovide	101	74	8	19		73.26%
📁 filestore	677	377	68	232		55.68%
📁 fuse	40	0	0	40		0.00%
📁 keystore	104	79	12	13		75.96%
📁 namesys	554	378	45	131		68.23%
📁 p2p	206	179	8	19		86.89%
📁 pin	825	497	78	250		60.24%
📁 plugin	276	139	24	113		50.36%
📁 repo	746	458	105	183		61.39%
📁 thirdparty	143	93	13	37		65.03%
📄 blocks/blockstoreutil/remove.go	48	40	4	4		83.33%
📄 tar/format.go	110	84	12	14		76.36%
Project Totals (146 files)	13,384	8,342	1,420	3,622		62.32%

Figure 11.5: Code Coverage

11.6.5 Developer Discussions

To investigate if developers discuss a lot about the possible technical debt we decided to look for a few keywords ("technical debt", "refactor") in Pull Requests and issues.

In terms of discussion in issues, there are quite a few issues trying to discuss more specialized problems. As an example, we can present the issue #4843 which discusses a technical debt in the CLI. Another example could be the issue #5723 focusing on testing and refactoring of the block exchange protocol.

In terms of discussions in the code, we observed 78 TODO and 9 FIXME comments in the source code.

11.6.6 Technical Debt Evolution

To observe technical debt evolution, we decided to run SonarQube on the latest release version and a few older versions from the past three years.

version	technical debt	issues	TODOs	Duplications	LOC
0.4.19 (02.2019)	4d 3h	325	97	1.1%, 18 blocks	27k
0.4.17 (08.2018)	3d 7h	356	93	1.6%, 54 blocks	36k
0.4.14 (03.2018)	4d 2h	353	186	1.7%, 59 blocks	36k
0.4.10 (07.2017)	4d	374	152	1.3%, 61 blocks	37k
0.4.2 (05.2016)	8d	541	184	1.3%, 73 blocks	58k

By analyzing the basic values we can see that the technical debt is fluctuating around 4 days. It is also clear that in the last year architects decided to extract some modules outside of the project as we could observe a significant loss in LOC between 0.4.17 and 0.4.19 versions. The number of TODOs, issues and duplications reduced throughout the versions, which indicates that the system is slowly getting more mature.

11.6.7 Conclusions to Technical Debt

From our research of technical debt we noticed that developers are trying their best to improve quality of the code, they discuss a lot on almost every GitHub issue before implementing the change, but also aim to increase code readability and understanding. In general, technical debt in terms of code quality is not remarkable, however, documentation debt is significant. We observed that there is still quite a lot of functions to be covered by the tests. As our remedies for the code we would consider:

- improve documentation and add the requirement to contribution guide to enforce documenting the code
- reduce Cognitive Complexity in longest functions as one can get lost while trying to understand or correct the code
- cover more blocks with tests
- improve tests implementation (e.g. real comparison, not only checking if it throws the exception)
- adding more constants instead of duplicating the same parameter in various places
- revisit duplicated blocks
- remove unused parameters where it is possible to improve code readability

11.7   Development View

In this section we explore the architecture of go-ipfs from the development viewpoint. We discuss mainly the package structure organization, responsibilities of different modules, dependency management, and testing approach.

11.7.1   Package Hierarchy

The whole project is separated into many reusable components. The following diagram gives an overview of the main packages, shows dependencies between them, and groups packages by layers of the IPFS stack. The go-ipfs repository then serves as the main package which wires all components together, allows to run an IPFS node daemon, and implements the CLI.

11.7.2   Network

The network layer handles point-to-point communication between nodes in the network and is based on *NAT traversal* techniques like *hole punching*, *port mapping*, and *relay*. It supports multiple transports as TCP, UDP, or WebSockets. The network layer is implemented in the libp2p library.

11.7.3   Routing

The routing layer server two purposes. First, it provides **peer routing** to help find other nodes in the network. An initial peer discovery is performed by connecting to a configurable bootstrap list of reliable nodes, which are in turn used to discover other peers.

Secondly, it provides **content routing** to find data published on IPFS. The current implementation is based on a *Distributed Hash Table (DHT)* protocol **Kademlia**, which is also used for example in BitTorrent.

11.7.4   Block Exchange

Data exchange in the IPFS protocol happens by exchanging data blocks using a BitTorrent-inspired protocol BitSwap, which incentivizes block storage and exchange. Each node has a list of blocks it maintains and a list of blocks it wants to acquire. Furthermore, each pair of nodes has a ledger maintaining number of bytes uploaded and downloaded. As the ratio of downloaded to uploaded bytes for a node increases, it is likely that other node will refuse to serve blocks until the debt ratio improves.

11.7.5   Merkledag

The **Merkledag** is the main data structure for representing files in IPFS. It is a *directed acyclic graph (DAG)* whose edges are hashes and nodes represent data corresponding to those hashes. IPFS is content-addressable, which means that each file can be accessed by knowing its hash. This allows for content deduplication, as multiple files with the same content are physically only stored once thanks to the fact they have the same hash. Git takes advantage of the same principles for efficient file versioning.
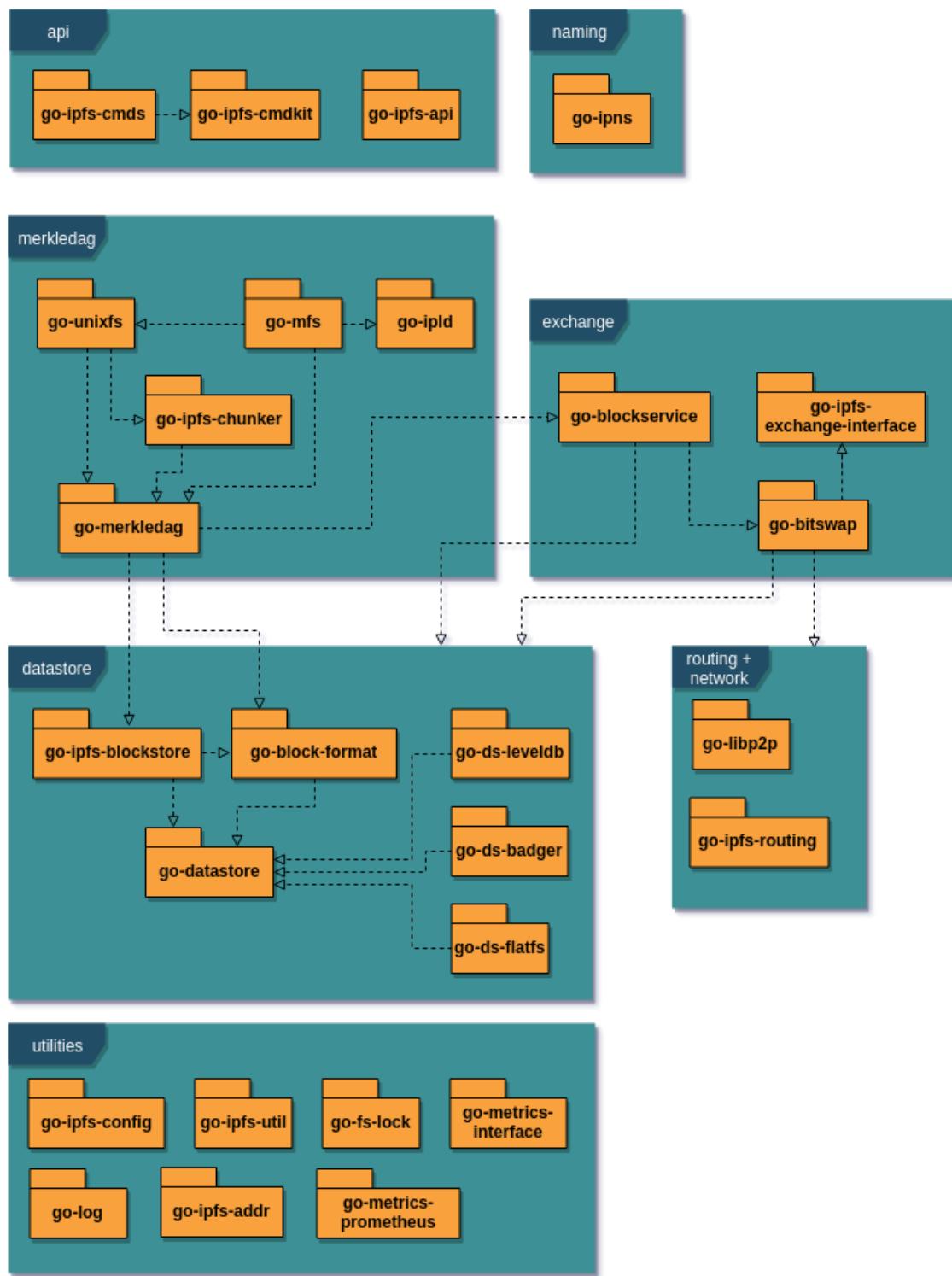
Figure 11.6: Package Diagram

Files are addressed by multihashes, a self-describing hash format. A multihash is composed of a hash algorithm name, the length of the hash and the hash itself. This allows to easily upgrade to a new hashing algorithm in future while maintaining backwards compatibility.

IPFS currently allows blocks up to 1 MiB. This means files larger than this have to be split by a chunker into small chunks. Those chunks are used to build a DAG.

In order to add the functionalities expected from a fully-fledged filesystem, another layer of abstraction is added with UnixFS, which specifies certain meta data to add features such as symlinks and directories.

While immutability is inherent in the file structure of IPFS, it is not always wanted. For mutability, another abstraction layer, Mutable File System (MFS), is added. In UnixFS, files and directories are addressed by the hash of the data contained within the files. In MFS, this is abstracted away behind human readable names.

11.7.6   Data Store

The persistant block storage is provided by datastores. Besides the default flatfs implementation which uses sharded directories and flat files, there are also experimental datastores using BadgerDB and LevelDB key-value databases with better performance.

11.7.7   Naming

The content-addressable data are by their nature immutable. Whenever the content is changed, its hash and thus the address also changes. To allow modifying shared data without need to distribute the new address each time the content is changed, there is a naming layer called **InterPlanetary Name System (IPNS)**. Every node has a public-private key pair, and names in IPNS are simply hashes of public keys. A node can assign an IPFS hash to its IPNS address and distribute the record using the IPFS routing system. The idea is based on the *Self-certifying File System (SFS)* [7].

11.8   CLI Architecture

There are several interfaces that can be used to communicate with an IPFS node. First, the user can communicate with the IPFS daemon via an HTTP API. Secondly, the user can use the CLI, which is able to perform some commands locally, but for online commands it requires a daemon with which it communicates over the HTTP API. Finally, application developers can include the go-ipfs package as a dependency and use its Core API.

The architecture of the CLI is shown in the following diagram. First, *CLI Parser* parses the command and its arguments and forwards it to the *CLI Dispatcher*. Each command is defined in a separate `Command` struct implementing a `Run` function which makes required calls either via Core API to the IPFS node directly, or via the HTTP API to the IPFS daemon in case the daemon is running. The command can also implement a `PreRun` hook to preprocess the user input and `PostRun` to format the output.

---

[7]https://en.wikipedia.org/wiki/Self-certifying_File_System

The structure of the daemon is quite similar, apart from the input handling, to the structure of the CLI. The HTTP-API server accept incoming calls, which are dispatched and called against the IPFS node.
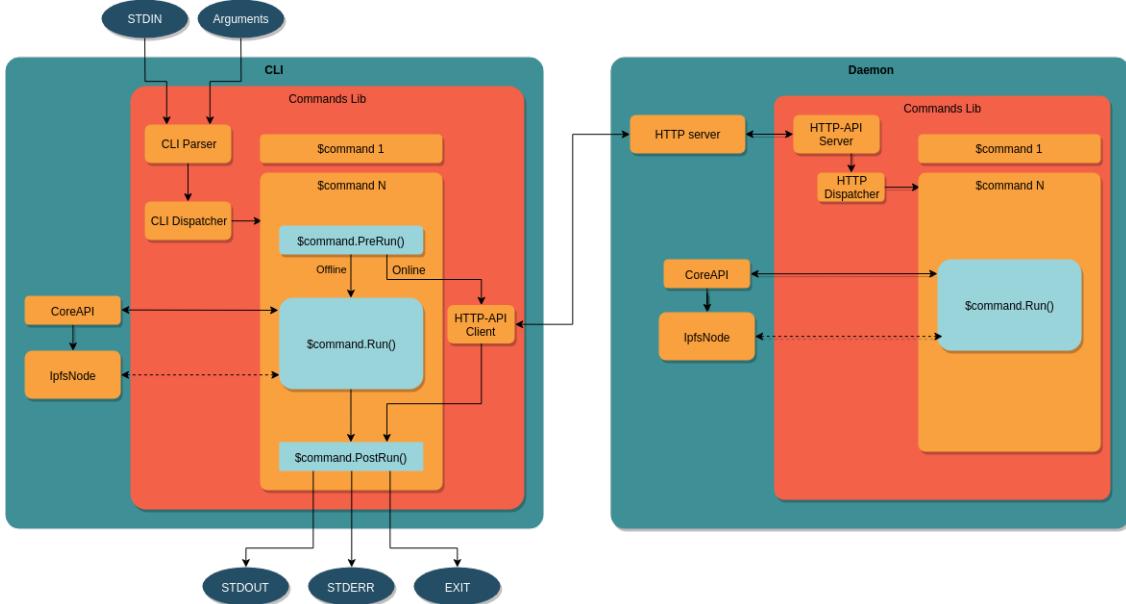
Figure 11.7: CLI Architecture Diagram

11.9   Dependency Management

As opposed to a monorepo [8] strategy adopted by many companies, IPFS tries to maintain modular architecture by developing each larger package in a separate Git repository. While this promotes reusability and enforces strict separation of concerns, it comes with more difficult development and dependency management if multiple components need to be changed at the same time.

Until recently, Go tool did not allow to depend on specific versions of external packages. Therefore, IPFS team came up with their own package management tool gx, which allows to publish versioned packages on IPFS and in turn depend on the specific version of a package by specifying its hash in `package.json`.

Since introduction of Go 1.11 in August 2018, there is an official support for versioned modules. [9] As it was not sustainable to maintain both systems, IPFS is now in the process of switching over to go modules and dropping gx support. [10]

---

[8]Monorepo is a development strategy where the whole codebase is stored in a single monolithic repository
[9]https://github.com/golang/go/wiki/Modules
[10]https://github.com/ipfs/go-ipfs/issues/5850

11.10   Testing

There are two types of tests in the project. Firstly, traditional go unit tests are stored in files with `_test.go` suffix just next to the source code files in their respective packages.

Integration tests located in the `/test` directory are bash scripts written with help of Sharness, a shell library for testing Unix programs. It is inspired by the test suite of the Git project, and it is developed by an IPFS developer @chriscool.

All tests are run automatically by CircleCI for every Pull Request.

11.11   Deployment View

In this section we explore IPFS from the deployment perspective, specify supported platforms, hardware requirements and runtime dependencies.

At the base of any IPFS deployment will be the IPFS daemon. The IPFS daemon is the base node that communicates with other nodes using p2p protocols in order to participate in the global IPFS network. The daemon may add, aquire, and serve content for the network, or for the clients that are dependent on it.

11.11.1   Installation

Release versions of go-ipfs come in prebuilt packages for all supported platforms and can be downloaded from the distributions page, which is hosted on IPFS itself. It can also be installed using package managers specific for target platforms. Another way is to install using the ipfs-update tool, which also provides a convenient way to update the software and revert to the previous version in case the newly installed version has any issues.

11.11.2   System Requirements

IPFS can run on macOS, FreeBSD, Linux and Windows. There are prebuilt packages for i386 and amd64 CPU architectures for all systems. On Linux, arm and arm64 architecture is supported as well. The program requires at least 1 MB RAM for stable runtime, though 2 MB RAM is recommended. [11]

11.11.3   Network Requirements

The daemon uses port `4001` to communicate with other peers in the network. The gateway which is used to access the IPFS content is running on port `8080` by default. Therefore, the firewall should be properly configured so that those ports are open for the TCP protocol.

There is also a REST API running on the port `5001`. However, that one should not be made accessible from the public network. This would be considered a vulnerability as anyone would be able to change the node configuration remotely.

---

[11]https://github.com/ipfs/ipfs-update

11.11.4 FUSE

For daemon to be able to mount to `/ipfs` or `/ipns` namespaces, FUSE (Filesystem in Userspace) needs to be installed. FUSE support varies across different operating systems. The installation on Linux is straightforward, as `fuse` should be available in most package managers. On macOS, it is required to use `osxfuse >= 2.7.0`, as it has been discovered that previous versions cause a kernel panic. For that reason, the IPFS daemon checks the osxfuse version and prevents mounting on older versions. Mounting is currently not supported on Windows. [12]

11.11.5 IPFS Cluster

For large-scale IPFS deployments, IPFS cluster, a pinset orchestration tool, can be utilized. It consists of a cluster service that should be run on the server along with go-ipfs, and a client CLI application which helps to manage pins across a large cluster of IPFS daemons.

11.12 Conclusion

This chapter gives an overview of the IPFS project and particularly the architecture of its Go implementation. Firstly, we identified different stakeholders interested in the project. To better understand the structure, relationships, and dependencies, we created the Context and Development View. We also examined the technical debt of the project in terms of code quality and testing perspective. Finally, we analyzed the requirements for deployment environment in the Deployment View.

We noticed that the project is very modularized to make it reusable and easy to change, however, in the beginning, this makes it a bit difficult to understand how all pieces fit together.

The team is putting great efforts to maintain a good quality of the code. There is also a great number of discussions about possible improvements in terms of the current state of the project and team operation. Nonetheless, we found that the lack of code documentation and low unit test coverage make it difficult to understand the inner workings of the code for new contributors. This oversight is probably caused by the fact that the project is still in an alpha version and evolves quickly.

Overall, we are very optimistic with the direction of the project and hope it will help to shape the future where the Web is decentralized again, as it was supposed to be from the beginning.

---

[12]https://github.com/ipfs/go-ipfs/issues/5003

Chapter 12

Keras: Fast Neural Network Experimentation

12.1   Contents

12.2   Introduction

Keras is an open-source library which has the aim to enable fast neural networks experimentation [9]. Among the implementations that Keras provides, there can be mentioned neural networks layers, activation functions and optimizers.

Initially developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), its initial release took place on 27th of March 2015. Starting from 2017, Keras was acquired by Google's TensorFlow team. The latest version (2.2.4) of Keras has been available since 3 October 2018 [10].

12.3   Stakeholders

In order to determine the stakeholders of the system, we have analyzed 20 of the most commented pull requests from the Keras GitHub page [11], both merged and unmerged. This helped us identify key roles in the decision making process, as well as the main factors when it comes to reaching a decision. We will further discuss the different types of stakeholders identified in this process, categorized based on the description provided by Rozanski and Woods [1]. Moreover, we identified and presented additional stakeholders categories, relevant for Keras.

12.3.1   Acquirers

They have the responsibility of representing the commercial interests of the product or system and oversee its development [1]. Keras had its initial release on 27 March 2015 by Francois Chollet, a Google engineer. In 2017 Keras received permanent support from Google's TensorFlow team, which made Google one of the most important acquirers. Apart from Google, the other acquire is Francois Chollet, the founder of Keras [5].

12.3.2   Assessors

They have the responsibility to ensure that the system respects the standards and legal regulations [1]. Keras operates under the MIT license [12]. The copyrights of the contributions made by each entity stay theirs. No clear stakeholder with the role of assessor was identified.

12.3.3   Communicators

They have the responsibility to provide a useful description of the system for the other stakeholders [1]. Any contributors who added a new feature are communicators, since they have to demonstrate its functionality and document it. The main communicator for Keras is Francois Chollet [5], the one who gives the public the latest informations about its development.

12.3.4   Developers

They are responsible for the construction of the system [1]. For Keras, the developers are the contributors from GitHub represented by the Keras Team (Francois Chollet (@fchollet the founder of Keras), Fariz Rahman (@farizrahman4u), Taehoon Lee (@taehoonlee), Gabriel de Marmiesse (@gabrieldemarmiesse)) and the Open Source Community.

12.3.5   Maintainers

Their role is to manage the evolution of the system after it has been deployed and is being operated [1]. Depending on the timeline (since acquired by Google), Keras had different teams contributing to its maintenance. The owner, Francois Chollet [5], has always been part of the maintenance cycle, but Open

Source contributors and the Keras Team organization always support the cycle by making reviews of the pull requests.

12.3.6 Production engineers

Their role is to provide a test environment for the newly added functionalities [1]. Furthermore, they need to manage the infrastructure of the software and the future release versions of it. Since the latter was already tackled in our identification of the system administrators stakeholders, we only focused on the first specified role. We identify Francois Chollet [5] as the production engineer, as he is in charge of defining the testing environment.

12.3.7 Suppliers

They provide the resources for the system to operate (hardware, software or infrastructure) [1]. In the case of Keras, the suppliers are represented by:

- Google [13]: Backs the development
- Microsoft [14]: Maintains CNTK backend
- Amazon AWS [15]: Maintains Keras fork with MXNet support
- Apple [16]: Provides CoreML
- TravisCI [17]: Testing and Continuous Integration
- MkDocs [18]: Documentation generation
- GitHub [19]: Hosting

12.3.8 Support Staff

Once the system is running, the support staff have the role of offering support to the users of the system [1]. In this case, this task is done by the Open Source Community and the Keras-team through different channels (for example Google forum, GitHub, Stackoverflow, etc.).

12.3.9 System administrators

Their role is to keep running the system after the deployment has been done [1]. Since Keras is an open source library, the users are the system administrators.

12.3.10 Testers

The role of the testers is to assess whether the quality of the system meets the requirements for deployment and usage [1]. The tests of the system are performed by the Keras-Team, as well as by the Open Source Community. Every pull request needs tests packed with the code, according to the contribution regulations [3].

12.3.11   Users

Their role is to assess whether the quality of the system meets the requirements for deployment and usage [1]. Since Keras is part of the Open Source software, any person interested in working on a deep learning project can use it. Additionally, users can be both industries and the research community. Some companies using Keras are, for example, Netflix, Uber or startups [2]. Keras is also used among the research community and research organizations, such as CERN and NASA [2].

12.3.12   Additional Stakeholders

12.3.12.1   Competitors

Their aim is to provide similar systems on the market. The main competitor for Keras is Gluon from Amazon [20], which is also a deep learning interface that supports Apache MXNet and Microsoft Cognitive Toolkit. Other alternatives include Deeplearning4j, Floyd, Mocha or Apache Spark. However, Keras is leading when it comes to the number of users.

12.3.12.2   Enthusiasts

People who help other users and have knowledge about the product. They introduce or help others to use the product and they can be identified on Q&A websites.

12.3.12.3   Media

The media is represented by articles on websites about Keras. Some examples that include Keras are "Deep learning in KNIME analytics platform" posted on IT Portal or "What will be the Growth of Deep Learning Software Market?" posted on Digital Journal.

12.3.12.4   Facilitators

The facilitators aim to teach users how to use Keras in different applications. They offer courses or tutorials and benefit from the existance of Keras, thus the evolution of the library directly influences them as they need to keep their materials up to date. On Udemy there are several courses about Keras, the highest ranked ones being offered by Lazy Programmer Inc., Jose Portilla and Francesco Mosconi.

12.3.13   Integrators

Integrators are persons responsible for checking the pull requests and ensuring a proper evolution of the software through the added functionalities.

In order to identify the integrators for Keras, we checked the commits to the master branch [11]. We have identified four main integrators responsible for committing the different functionalities authored by the open source community:

- Francois Chollet: @fchollet
- Fariz Rahman: @farizrahman4u
- Taehoon Lee: @taehoonlee
- Gabriel de Marmiesse: @gabrieldemarmiesse

Although they are all responsible with testing the functionalities proposed in the pull requests and making sure the code base is clean and the coding style is respected, Francois Chollet is the one always performing the final merge.

12.3.14 Contacted Persons

Before starting to develop the architecture analysis of Keras, we have contacted Francois Chollet [5], the project owner, via email. We asked whether he could provide any documentation regarding the development, design and architecture processes of Keras. We contacted him based on the details we found on his GitHub account [5]. Unfortunately, we did not receive any reply from him.

When further researching the Keras documentation [6] we have identified possible means for asking questions, where the Keras team and the open source community can answer, namely the Keras Google group [7] and the Keras Slack channel [8].

12.3.15 Pull Requests Theory

In the general sense, the decision regarding a pull request starts with the evaluation of the proposed functionality or improvement in the context of the product. If this evaluation does not pass, the developers simply reach the decision of not moving ahead. In case the proposed functionality makes sense, the main factors that influence the incorporation are the compliance to the defined Contributing Guidelines specified in the Keras documentation and passing the reviewing process. On the other hand, the development team decides to not merge the pull request if there is an overlap with already existing functionality or the code is not of high quality. Additionally, there are cases when the authors do not finish the change proposed in the pull request.

For the exact list of the analyzed pull requests, refer to the Appendix.

12.3.16 Power-Interest grid

The power-interest grid is a graphical representation used for classifying the stakeholders [21]. The four categories presented are:

Keep Informed corresponds to stakeholders that are very interested in the evolution of the project. These people do not have much power on the product, but they are directly affected by its development. An example of these types of stakeholders is represented by competitors. Gluon, Amazon's library for deep learning, needs to be informed by the functionalities of Keras because it should provide the same (or better) quality of their system.

Keep Satisfied corresponds to stakeholders with a high amount of power on a project. These people are usually interested in the progress of the project. In our case, as we mentioned that Microsoft and Nvidia

are supporting Keras, they are concerned of the success of the product. However, they do not need to be informed about all the updates that are made on Keras.

Monitor corresponds to stakeholders that are not very interested in the evolution of Keras and do not have power on the product. From these stakeholders, we can mention the media.

Manage Closely corresponds to the most important category of stakeholders. Their poor management could easily result in the failure of the product. Acquirers, Google and Francois Chollet, are the most representative stakeholders from this category due to their strong direct impact on the project.

12.4   Context View

The main purpose of the context view is to determine the connections between Keras and external entities [1]. Thus it could also be possible to identify any threats or vulnerabilities of the system from components that it is linked to.

12.4.0.1   System scope and responsibilities

Keras is defined as a high-level neural network API, written in Python and capable of running on top of TensorFlow, Theano and CNTK [6]. Their focus is to allow fast experimentation.

The system allows:

1. Easy and fast prototyping
2. Running models on CPU and GPU devices
3. Support of convolutional and recurrent neural networks
4. A way to define complex models if necessary with the **Keras Functional API**
5. Flexilibilty between banckends to be used

The system does not allow:

1. Provide an end to end functionality, it is an interface
2. Provide advanced and specific operations dependent on the back end to be used

12.4.1   Diagram

In the figure below, the context diagram is shown, capturing the relationship between the system and its environment. We have identified ten external entities, each having different connections with the system.

Keras is written in Python, originally by Francois Chollet. Currently, it is being developed by various entities, mainly the Keras Team organization from GitHub and the Open Source Community. It can run both on CPU and GPU and it supports Microsoft's CNTK, Tensorflow and Theano. It is managed on GitHub and when it comes to integration and testing, Travis CI is used.

The users cover a wide range, from research organizations to industries. The support is provided mainly through the documentation, that is generated using MKDocs, or through several communication channels, namely Slack, Google Groups or StackOverflow. Concerning dependencies, we have identified both
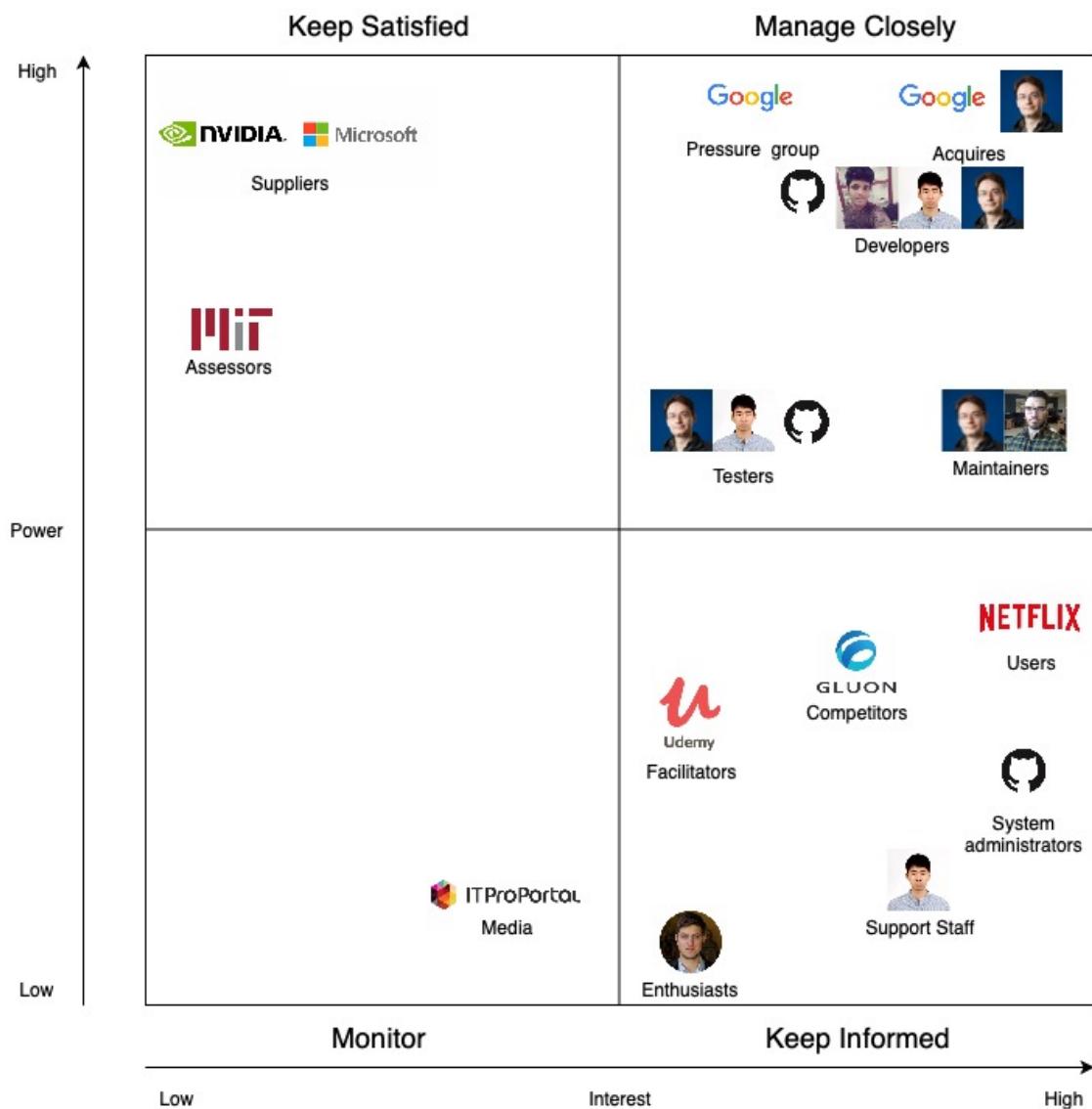
Figure 12.1: GitHub Logo
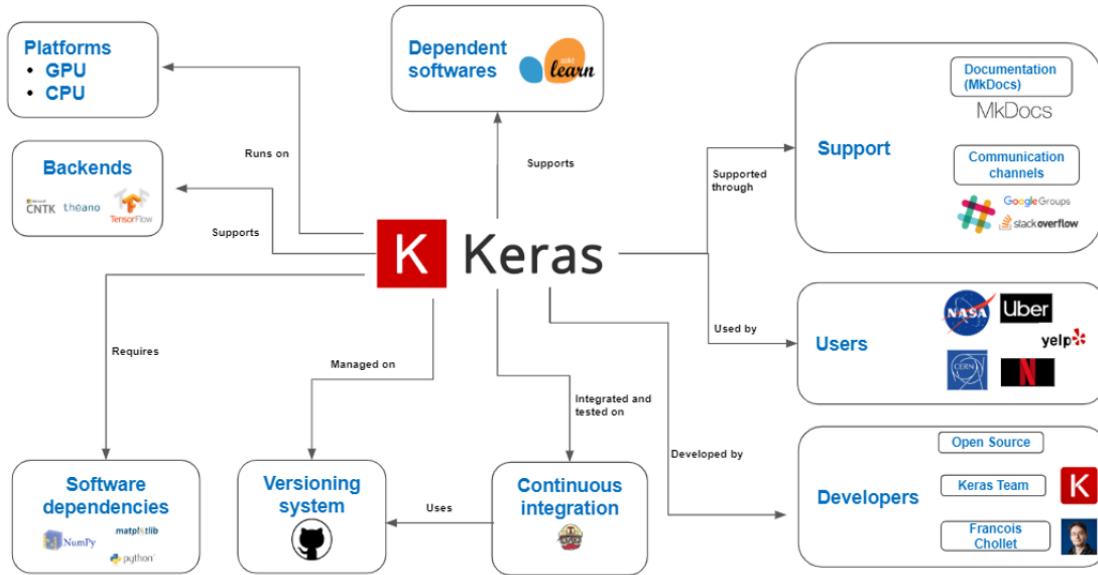
Figure 12.2: GitHub Logo

dependent software, supported by Keras (Scikit-learn), as well as software dependencies that Keras requires (Python, NumPy, matplotlib).

12.5   Development View

The development view of a system describes the architecture that supports the development process [1]. In this section we will describe the various aspects composing the overall process.

12.5.1   Module Structure

The Module Structure for Keras is visualized in the figure below.

The modules visualized in the figure above have the following functionalities:

- **Core**: Includes the main components that implement the essential functionalities for defining Deep Learning model architectures. These include the definition of different types of layers, the definition of the topology of the networks and the possible backends for executing the generated models.

- **Auxiliary Modules**: Provide additional functionalities to complement the implementation of the Deep Learning models. These include preprocessing components for both the data (such as image rotations or shifts) and the models created through the Core module (such as normalizing, counting the number of parameters or printing the summary of the architecture). Moreover, several datasets are provided (for example, CIFAR-10 or MNIST).
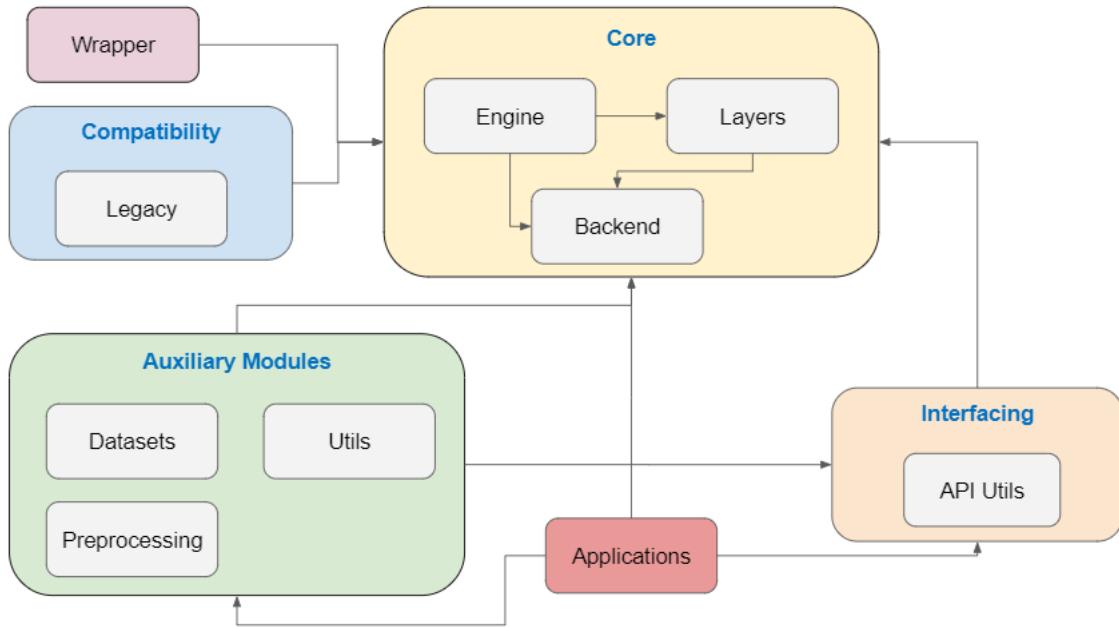
Figure 12.3: GitHub Logo

- **Wrapper**: Provides a wrapper for scikit-learn [22].

- **Compatibility**: Provides support for previous versions of Keras.

- **Applications**: Provide implementations of well-known architectures of pretrained models, such as VGG16 or Inception_v3.

- **Interfacing**: Provides an API to the Core functionalities.

12.5.2   Codeline Organization

12.5.2.1   Source Code Structure

The structure of the source code for Keras is summarized in the figure below.

We have identified a logical structure of the Keras code in four different components:

- Functionality
- Installation and deployment
- Documentation
- Testing

The **Functionality** component contains the code that provides the core functionality for the product. It focuses on implementing the architectural components of a Deep Learning model, it provides wrappers for other platforms and ensures the integration with several backends.
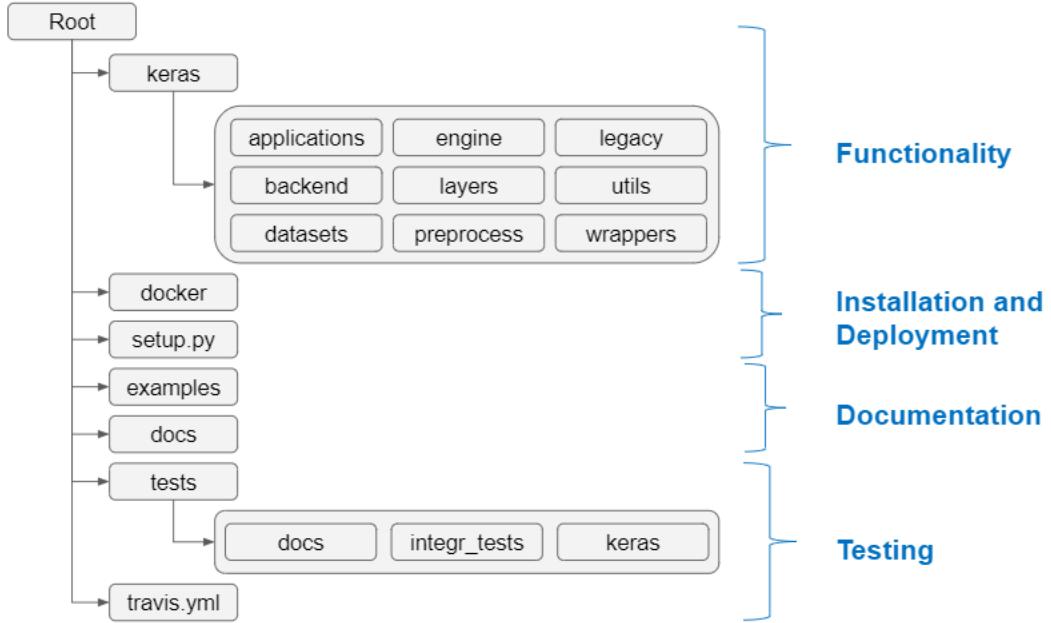
Figure 12.4: GitHub Logo

The **Installation and Deployment** component contains the necessary scripts and configurations to perform releases and installations.

The **Documenation** component contains the files necessary for generating the documentation of Keras. It is based on two directories, namely `docs` and `examples`. `Docs` contains the code for generating the documentation visible on the Keras website, whereas `examples` includes examples of using Keras.

The **Testing** component includes scripts that are responsible for testing the functionality of each individual component in the Keras directory and the configuration of the continuous integration platform for performing the autonomous tests through TravisCI.

12.5.2.2   Build, Integration, and Test Approach

The approach Keras uses for integrating new functionalities into the product is described in the contribution policies on their GitHub repository [11]. Any person interested in creating a pull request should write a design document for motivating the proposed functionality. After the functionality is approved, the code can be written and accompanied by proper docstrings. Moreover, the tests have to be defined and run locally on all the supported backends before finalizing the pull request.

After the pull request is made, the automated tests are run in the continuous integration platform, TravisCI [17], and checked by a person that Francois Chollet [5], the project owner, assigns. If everything runs correctly, then the PR is merged by Team Keras and tests are run again through the continuous integration platform.

It is important to mention that the configuration for the automated tests run through TravisCI is included in the GitHub repository of Keras [11], more specifically in the `travis.yml` file.

This process can be visualized in the figure below, for the Pull Request #12500 ("Revise 'logsumexp' in np backend and reduce those redundant tests").

Figure 12.5: GitHub Logo

12.5.2.3 Release Process

Through our analysis, we identified that a first step in the release process is preparing the release, as the procedure is not automated. The preparation includes changing the Keras version and the download URL for accessing the latest available version.

Moreover, we identified two types of releases, depending on the size of the changes they make:

- Periodic releases: they happen approximately every month and they focus on issues such as bug fixes, API changes, performance or documentation improvements.
- Major releases: they happen when significant, long time changes are made, that affect the overall functionality and structure of the projects. These include, for example, adding completely new functionalities, major changes to existing functionalities or version changes.

The releases are performed and documented [23] by the owner of the project, Francois Chollet [5].

12.5.3   Common Processing Elements

- **Initialization**: each package of Keras contains an `__init__. py` file in which there are imported all the packages needed by that specific module. This is done in order to avoid importing the same package in multiple files [3].
- **Logging**: the logging is done inside the Utils module by using `print_function` from `__future__` module. It makes use of the print function to display messages on the console [3].
- **Internationalization**: this element is not that obvious in Keras. All user and administrator visible strings are hard-coded in the source code. Moreover, they provide translated documentation in three major Asian languages: Japanese, Chinese and Korean [3].
- **Third party libraries**: Keras relies mostly on its own libraries. However, they also use some core libraries, such as *numpy* and they defined a wrapper for *Scikit-Learn* [3].
- **Configuration parameters**: at startup, the configuration is done locally by assigning values for some global variables. E.g. `_FLOATX = 'float32'`, which specifies the type of float to use throughout the session. Furthermore, Keras uses the file `savings.py` for saving and loading the configuration of the models [3].

12.5.4   Instrumentation Analysis

Instrumentation is the act of inserting code for logging information about the current state of the system and the usage of resources [1]. It is used to monitor while debugging.

Keras does not have clear instrumentation. They make use of the `print_function` and `warning` module in order to display messages via the console. A powerful tool for including instrumentation is **Lantz** [24], a toolkit for automation and instrumentation in Python.

12.5.5   Standardization of Design

Keras provides for potential contributors a set of guidelines to follow [1]. These guidelines are specified in `CONTRIBUTING.md`. Moreover, they provide templates for standardized writing of issues and pull requests in `PULL_REQUEST_TEMPLATE.md` and `ISSUE_TEMPLATE.md`.

12.5.6   Code Conventions

There are two subcategories of PRs: improvements & bug-fixes and adding experimental features. The former category goes to the Keras **core** master branch, while the later goes to **keras-contrib**, unless the feature was requested [3].

In order to submit the changed code, the developer must fork the repository and make the changes on a feature branch.

Guidelines:

- Respect the Google docstring style and there should be sections for *Arguments*, *Returns* and *Raises*;
- Provide unit tests;
- Run the test suite of Keras;
- Respect PEP8 (add link) syntax conventions;
- Update the documentation if adding new functionality.

12.5.7   Standardization of Testing

Keras team developed a set of tests for all the files inside the modules. All the testing files reside inside the tests directory and have the following name conventions: either `test_filename.py` or `filename_test.py`.

Apart from the regular test set, Keras also provides an integration test set, which verifies how well different modules interact with each other.

The command `py.test tests/` is used to run the test suite locally and `py.test --pep8 -m pep8` is used for checking the syntax conventions [3].

Some of the PEP8 errors can be **automatically** fixed by running `autopep8 -i --select <errors> <FILENAME>`.

12.6   Technical Debt

12.6.1   Identifying Technical Debt

We chose to analyze the `engine` module due to its increased complexity and given that all the major functionalities of Keras are relying on it.

We used SonarQube [25] in order to determine the quality of the code on the Keras repository. The results can be visualized in the image below.

- **Bugs** analysis:

This issue refers to programming errors that can lead to business disruption [26]. From SonarQube we can observe that Keras has 8 bugs. On closer analysis of the code, 2 bugs are not influencing the quality of the code. They have been reported as unnecessary operations. The other 6 bugs are focused on the documentation.

- **Vulnerability** analysis:
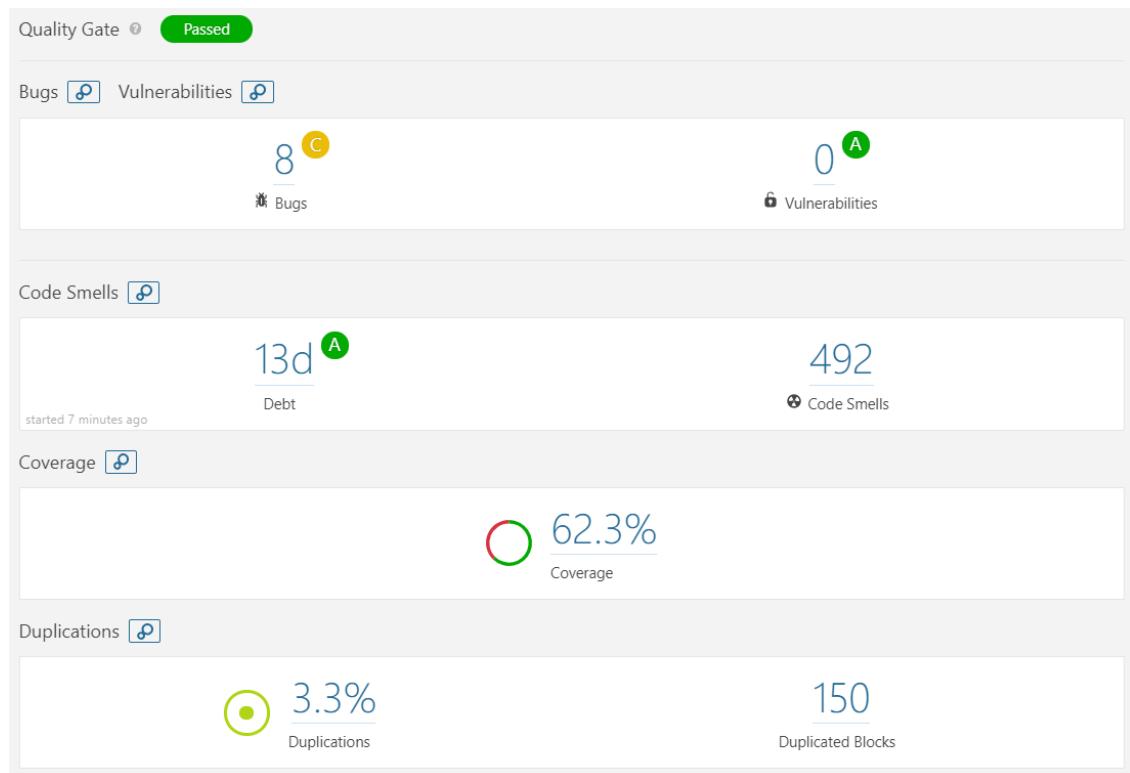
Figure 12.6: GitHub Logo

This issue refers to flaws in the programs that can lead to the usage of the application in a different way that it has been designed for [26]. No vulnerability was detected in the Keras code.

- **Code smells** analysis:

This analysis focuses on the maintainability issues, which is directly correlated to the ability, the cost and the time to make changes over time in the code base [26].

We can observe that the debt is 13 days. Considering the complexity and size of the Keras project, we consider this to be a good result. This is also observed from the `A` maintainability rating associated to it, which means that the technical debt ratio is less than 5%.

When it comes to the code smells, 492 are identified. These are grouped in three severity levels: critical (22.15%), major (47.15%) and minor (30.7%). Regarding the critical code smells, all of them propose refactoring certain functionalities for reducing the cognitive complexity. These refer to modules that are not directly involved in the core functionality of Keras, such as `docs`, `legacy` and `examples`, as well as more central modules, namely `backend`, `engine`, `layers` and `utils`. Concerning the major code smells, they mostly refer to merging `if` statements with the enclosing ones, removing code that is commented out, pointing out functions with too many parameters or renaming functions. They refer to all the module in the Keras repository. Finally, the minor code smells are present in most of the modules as well and they are concerned with renaming classes or parameters or removing unused variables.

- **Duplication** analysis:

The overall percentage of duplication of Keras is 3.3%. However, most of the duplications can be found in the tests and examples modules. On closer analysis of the example modules, most of the duplicated lines are preprocessing techniques or parameters initializations, which are required operations. When it comes to the `test` module, most of the duplicated lines have the purpose of building a neural network, which is a necessary operation.

12.6.1.1 SOLID Violations

12.6.1.1.1 The Single Responsibility Principle (SRP): We believe that this principle is mostly violated in `keras/keras/engine/network.py` by the class `Network`. This class models a neural network as a directed acyclic graph of layers for which the following functionalities were identified [27]:

- Retrieval the updates and the losses of the model;
- Calling the model on new inputs;
- Computing output tensors for new inputs;
- Saving layers/weights to files;
- Deserializing layers/weights from files.

The class consists of 1100 lines, 33 methods and many of them have a cognitive complexity above 30. Its extended functionality has the consequence of higher coupling. Decoupling such a large class may be a solution, but, given the complexity of neural networks architecture, this may lead to dependency issues.

12.6.1.1.2 The Open-Closed Principle (OCP): A clear example of a violation for this principle is in `keras/keras/engine/saving.py`. The file reunites all the utilities for saving a model and is not even organized as a class. Hence, for the scenario of adding a feature for saving on a new file format, the developer must modify the source code, while not having a possibility of extending it. As a solution, we

considered refactoring the file as an interface. In this way, adding a new file format for saving would only imply creating a new child class. Thus, classes like `Network`, which uses saving functionalities, remain closed for modification and the interface stays open for extension [27]. Our **proposed improvement** is shown in the image below.
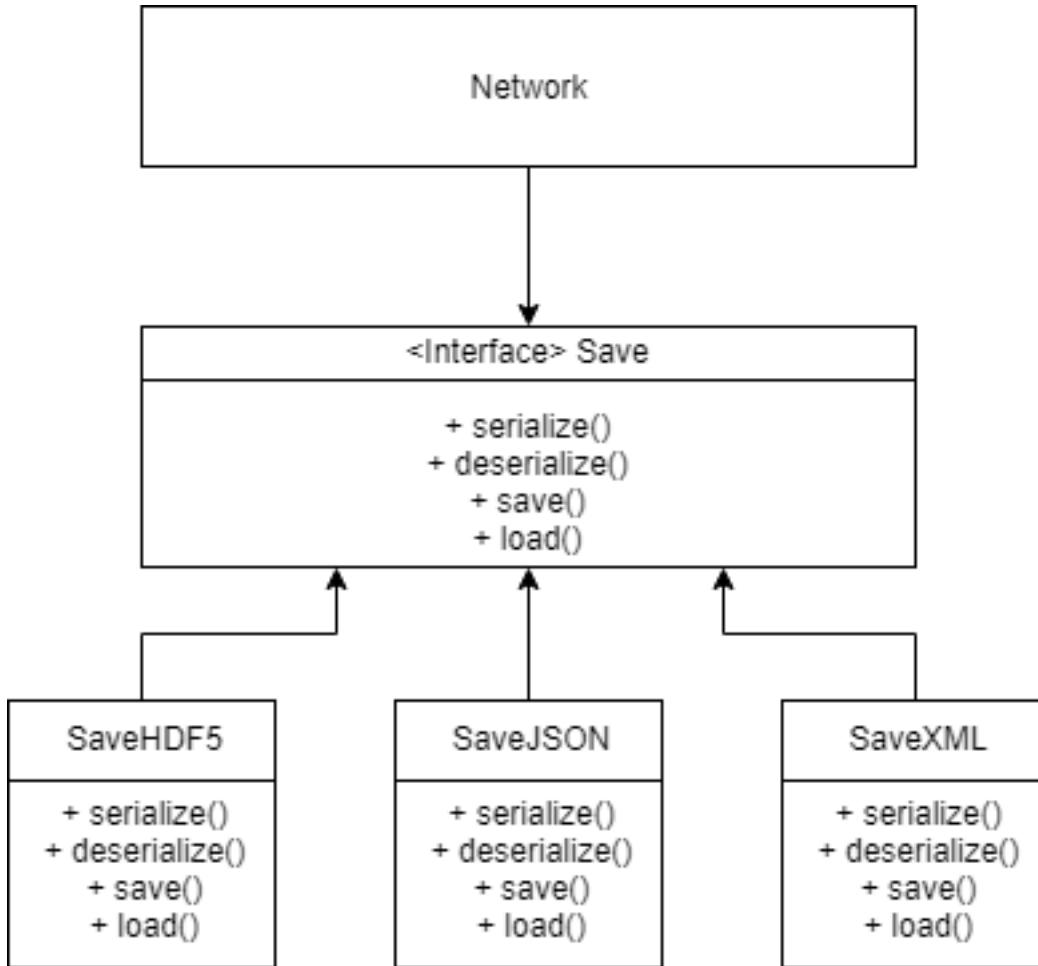
Figure 12.7: GitHub Logo

12.6.1.1.3   Liskov Substitution Principle (LSP):   Since the engine module of Keras is composed of a few classes, there are not many chances of violation for LSP. The inheritance tree is:

Each child class comes as an addition to the parent class and they mostly override methods for showing the configuration of the system. There were no casts found in the code, hence we believe that LSP is not violated.
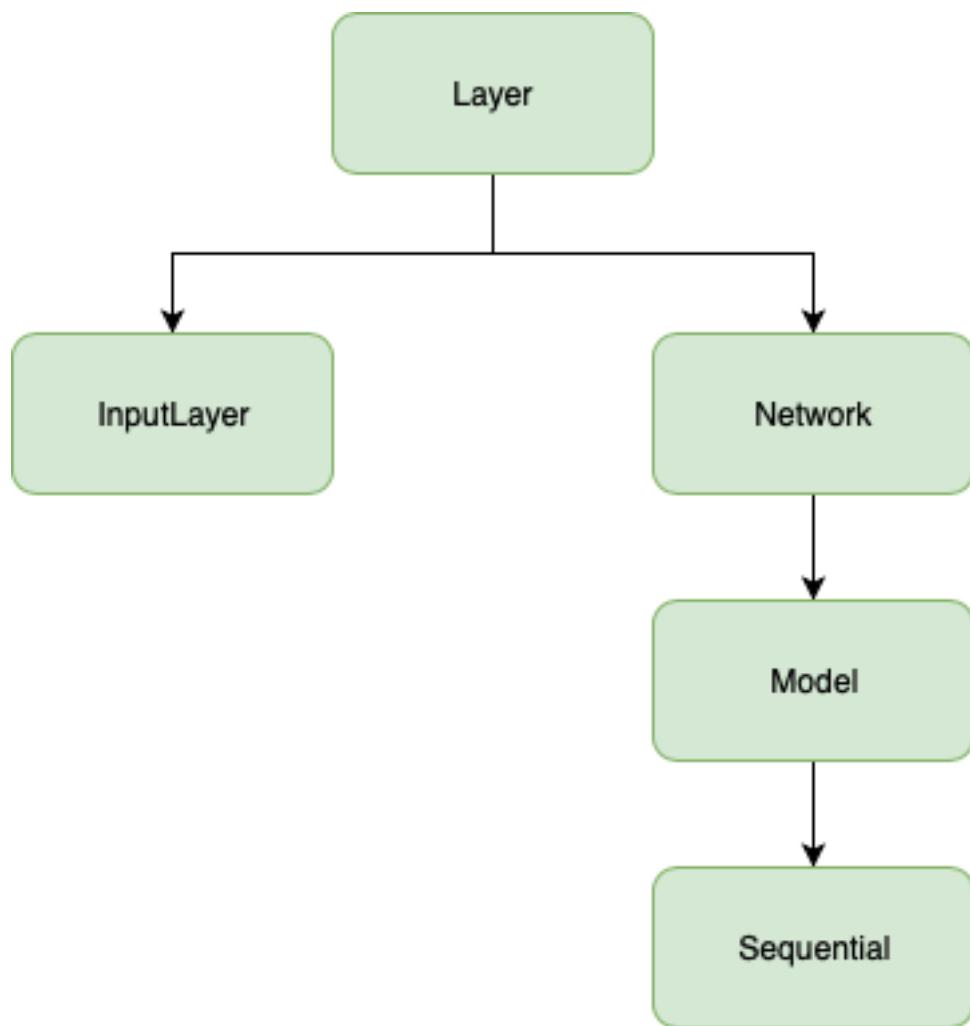
Figure 12.8: GitHub Logo

12.6.1.1.4   The Interface Segregation Principle (ISP):   Keras uses a legacy interface which contains converters for Keras 1 support in Keras 2. The interface is composed of 15 functions and expands on over 600 lines of code. This interface is used almost in every class from engine module, hence a change in it would require changes in the other classes. We believe that this violates ISP.

12.6.1.1.5   The Dependency Inversion Principle (DIP):   As can be observed from the inheritance tree, DIP is violated because the inheritance is done directly from the parent class and not through an interface for that respective class.

12.6.1.2   Discussions about Technical Debt

The developers of Keras communicate not only through git issues, but also through code comments (i.e. TODO). In order to better observe all these comments we used the command `git grep 'TODO' -- '*.py'` [28]. There are 21 TODOs. Some contain the developer assigned to solve them or the issue that they are addressing. Some examples include:

- `keras/engine/training.py: # Assumed tensor - TODO(fchollet) additional type check?`
- `tests/keras/engine/test_training.py: # TODO: resolve flakyness issue. Tracked with #11560`
- `keras/engine/training_utils.py: # TODO Dref360: Decide which pattern to follow. First needs a new TF Version`

Figure 12.9: GitHub Logo

Most of them either are about removing certain pieces of code or pointing to issues.

12.6.2   Identifying Testing Debt

12.6.2.1   Discussion of code coverage

The overall coverage of the `keras` module in the repository is 56%. The coverage per component is visualized in the figure below.

As we can observe, there is no coverage for: `applications`, `datasets`, `legacy` and `preprocessing`. Given the focus of Keras, we consider it reasonable that the `applications`, `preprocessing` and `datasets` modules do not have test coverage, as the first two reside in a different repository and the latter only facilitates access to data. However, when it comes to `legacy` we think that tests should be performed, since it ensures backwards compatibility.

Additionally, we observe very little test coverage when it comes to the `backend` component, which is a sensible decision since each backend is provided by third party entities, responsible of testing their functionalities.

Regarding the remaining components, the test coverage is higher, more specifically above 79%.

Overall, in our opinion the project is well-covered when it comes to testing, except for `legacy`, which we think should be tested.
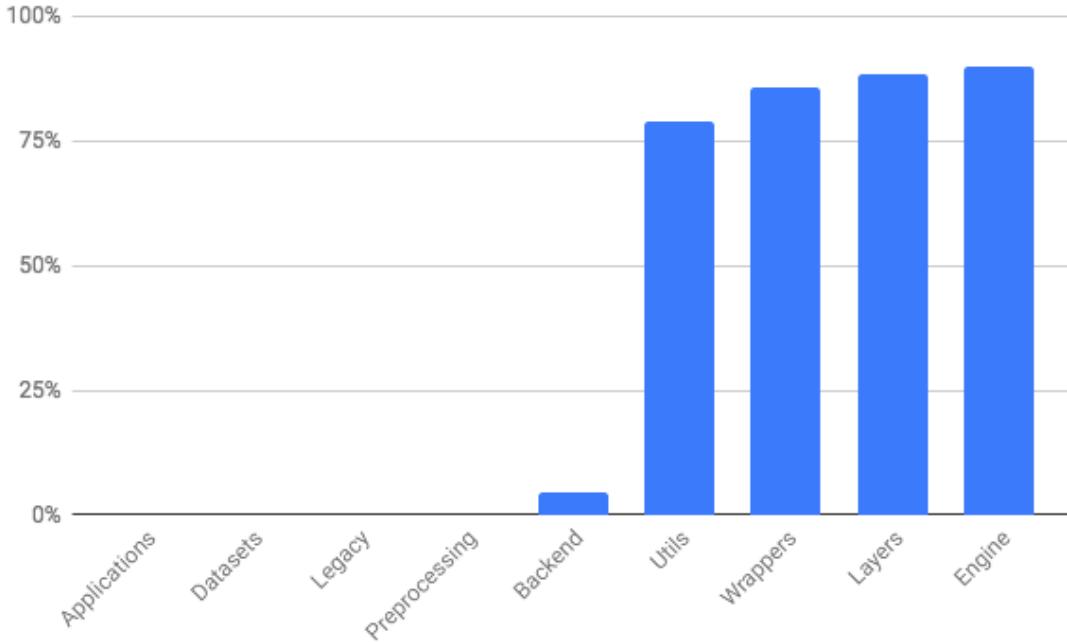
Figure 12.10: GitHub Logo

12.6.2.2 Discussion of testing procedure

Through additional analysis we have observed that Keras is tested using two procedures:

- Unit tests
- Integration tests

The coverage regarding the unit tests was already presented above. Concerning integration tests, these cover some of the modules omitted during the unit testing, more specifically `applications`, `datasets` and `preprocessing`. These are not included in the coverage analysis, since these modules are part of external projects and libraries.

Additionally, we have a remark. We have identified that each functionality of Keras depends on the backends. This fact results in partial testing, as, depending on the backend, some of the tests are skipped. This might affect the maintainability of the testing procedure. A code snippet with such an example can be seen below.


```
@pytest.mark.skipif((K.backend() == 'theano'),
                    reason='Bug with theano backend')
```

Figure 12.11: GitHub Logo

12.7   Deployment view

The main goal of this view is to define the physical environment in which the system is intended to run [1]. The case of Keras brings a broad set of possibilities because it is a front-end that facilitates the construction of Deep Learning models. Thus, we will look at two standard deployment architectures.

12.7.1   Dependencies

In order to install the latest version of Keras the following requirements need to be satisfied [6]:

- Python version should be 2.7-3.6
- Numpy Python library (>=1.9.1)
- Scipy Python library (>=0.14)
- Six Python library (>=1.9.0)
- Pyyaml
- H5py
- keras_application (==1.0.4)
- keras_preprocessing (1.0.2)

Keras installation can be realized by using the package installer available in Python (pip). The command for installing Keras is: `pip install Keras` This command automatically installs the required packages listed above.

12.7.2   Technology Compatibility

Before installing Keras, one of the three supported backends needs to be installed: Tensorflow, Theano, CNTK. However, team Keras recommends installing Tensorflow [6].

Depending on the application that needs to be developed, there are some additional optional requirements:

- cuDNN - this is recommended if the applications require running Keras on GPU
- HDF5 - this is recommended if the application requires running Keras models to disk
- graphviz and pydot - this is recommended for visualization purposes

Additionally, each of the backends has its own technical functional requirements [12], [13], [14], which we will summarize in the table below:

Resource

TensorFlow

Theano

CNTK

Operating Systems

Ubuntu 16.04 or later

Windows 7 or latermacOS 10.12.6 (Sierra) or later

Raspbian 9.0 or later

Linux (e.g. Ubuntu, CentOS6)

Windows

macOS

Windows (8.1 Pro, 10, Server2012 R2 Standard and later)

Linux (Ubuntu 16.04 LTS)

Compilers

GCC 4.8

g++ >=4.2

Visual Studio Enterprise 2017

12.7.3 Hardware Requirements

Any Keras deployment relies on the following requirements of hardware:

Resource

Specification

Processing

As part of the configurations needed, there can be the following types of processing:

CPU

4th, 5th, 6th, 7th, and 8th generation Intel(R) Core(TM) processor

Intel Xeon processor E5 v4 and v3 family

Intel Xeon Phi processor x205 product family (formerly Knights Mill)

Intel Atom(R) processor with Intel SSE4.1 support

```
      </li>
      <li>GPU:
        <ul>
            <li>NVIDIA (CUDA or cuDNN)</li>
            <li>OpenCL-enabled GPUs</li>
        </ul>
      </li>
      <li>Google TPUs: This applies only for the TensorFlow backend</li>
    </ul>
  </td>
</tr>
<tr>
  <th>Memory and disk</th>
  <td>Depends entirely on the Neural Network to be trained</td>
```

```
</tr>
<tr>
  <th>Optional</th>
  <td>Apache Spark clusters and Apache MLLib in the case that distributed training is necessary</td>
</tr>
```

12.7.3.1   Frontend platform models
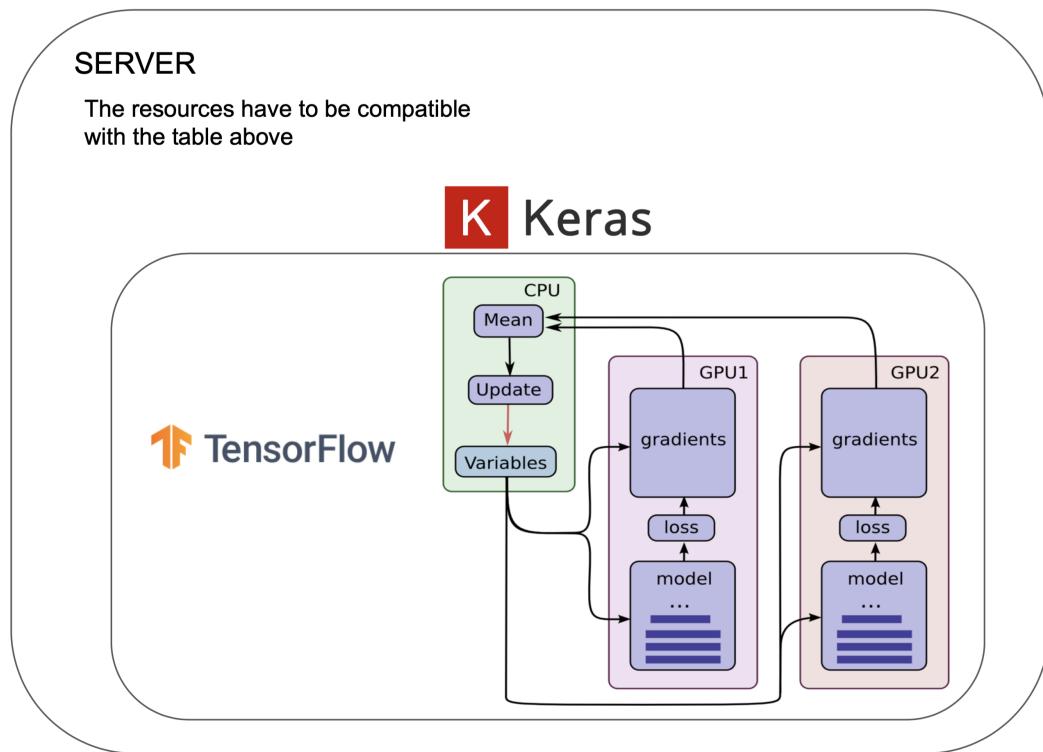
Figure 12.12: GitHub Logo

12.7.3.1.1   Multi-GPU supported by TensorFlow backend: In this case, Keras replicates a model and is executed on a dedicated GPU. The concatenation of the results is done on CPU into one big batch, returning a model fully trained at the end.

12.7.3.1.2   Distributed: Elephas implements a class of data-parallel algorithms on top of Keras, using Spark's RDDs and data frames. Keras Models are initialized on the driver, then serialized and shipped to workers (this is because Apache Spark is built on Scala), alongside with data and broadcasted model parameters. Spark workers deserialize the model, train their piece of data and send their gradients back to the driver.
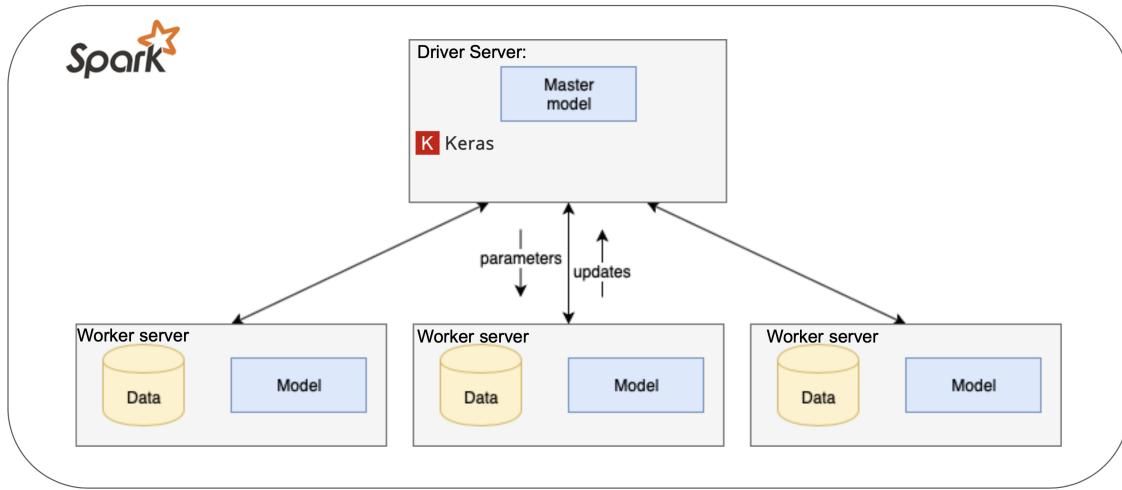
Figure 12.13: GitHub Logo

12.7.4 Historical Analysis

We created a summary of the historical evolution of Keras, presented in the table below, by analyzing the release notes for each new version release on the GitHub page [23]. We did not include each single version release, but we aggregated them in larger version releases (i.e. we aggregated versions 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4 in version 2.2).

Keras Version	Changes	Release Period
0.1	Preparations for Pypi release and Pypi release	June 2015 - August 2015
0.2	New Pypi release	October 2015
0.3	Modular Backends Introduction, New Pypi release	December 2015 - March 2016
1.0	Bug fixes and improvements on functionalities, New Pypi release	April 2016 - August 2016
1.1	New Pypi release	October 2016 - November 2016
1.2	New Pypi release	December 2016 - February 2017

Keras Version	Changes	Release Period
2.0	API changes; Deprecated layers removal (both convolutional layers and recurrent layers); New layers compatibilities; Variable name changes; Bug fixes and functionality improvements; Documentation improvements; CNTK backend addition; TensorBoard improvements; Coding style improvement; New features addition; Test coverage improvements; Incompatibilities solutions; Performances improvements; New Pypi release	May 2017- November 2017

2.1 | New APIs; Improvements to unit tests/CI.API; Performance and usability improvements; New models added in the 'applications' module; Improvements to example scripts; Documentation improvements; Bug fixes | November 2017 - April 2018

2.2 | Large refactors improving code structure and test time; Improvements to the documentation; API changes and improvements regarding usability; Performance improvements; Bug fixes | June 2018 - October 2018

12.8   Conclusions

In conclusion, Keras is very well structured and maintained software. We identified the most important stakeholder, Francois Chollet, which is in charge of establishing a clear plan of the project's development. Each pull request is closely analysed and tested in order to ensure good functionality. Furthermore, considering the modular nature of Keras, an extension can be easily performed.

In addition, from a code organization point of view, Keras' architecture is composed of various modules, each of them performing a different functionality. All of them are centred over a core module, the engine module, which is practically controlling the workflow. We consider that the fact that the modules' task does not overlap gives us an insight into the high performance of the software.

However, from the technical debt analysis, we observed that Keras also has some SOLID validation. As observed, these do not affect the project's functionality, but they might affect further expansion. Furthermore, we also believe that the legacy module should be provided with test coverage as it contains the main deep learning components.

12.9   References

1. Rozanski, N., & Woods, E. (2012). Software systems architecture: working with stakeholders using viewpoints and perspectives. Addison-Wesley.
2. Why use Keras: https://keras.io/why-use-keras/
3. Contribution guidelines of Keras: https://keras.io/contributing/
4. Robert C. Martin (2008). Clean Code.
5. Francois Chollet GitHub Profile: https://github.com/fchollet]

6. Keras Documentation: http://keras.io
7. Keras Google Group: https://groups.google.com/forum/#!forum/keras-users
8. Keras Slack Channel: https://kerasteam.slack.com/
9. Keras Website: https://keras.io/
10. Keras Wikipedia: https://en.wikipedia.org/wiki/Keras
11. Keras Github page: https://github.com/keras-team/keras
12. MIT License Keras: https://github.com/keras-team/keras/blob/master/LICENSE
13. Google Tensorflow: https://www.tensorflow.org/

14. Microsoft CNTK: https://docs.microsoft.com/en-us/cognitive-toolkit/
15. MXNet Support Amazon: https://aws.amazon.com/mxnet/
16. Apple CoreML: https://developer.apple.com/documentation/coreml
17. TravisCI: https://travis-ci.org/
18. MkDocs: https://www.mkdocs.org/
19. Github: https://github.com/
20. Amazon Gluon: https://aws.amazon.com/blogs/aws/introducing-gluon-a-new-library-for-machine-learning-from-aws-and-microsoft/
21. Power Interest grid: https://www.mindtools.com/pages/article/newPPM_07.htm
22. Scikit-learn: https://scikit-learn.org/stable/
23. Keras Releases: https://github.com/keras-team/keras/releases
24. Lantz:https://github.com/LabPy/lantz
25. SonarQube: https://www.sonarqube.org/
26. SonarSource: https://www.sonarsource.com/why-us/code-quality/
27. Robert C. Martin (2008). Clean Code.
28. Linux Command: https://www.cyberciti.biz/faq/howto-use-grep-command-in-linux-unix/

12.10 Appendix

The pull requests that we analyzed are the following:

Pull Request Number	Pull Request Title	Status
#244	Introduce time-masking to recurrent layers	Merged
#1623	Convolutional layers for 3D	Merged
#10047	MobileNetV2 keras implementation for imagenet weights for Tensorflow backend	Merged
#2152	Keras 1.0 preview	Merged
#9253	Add support for stateful metrics.	Merged
#6891	Fix the ordering bugs when using pickle_safe=True	Merged
#2523	Faster LSTM	Merged
#893	Lambda Layer	Merged
#2413	Support for masking in merged layers	Merged
#7980	Recurrent Attention API: Support constants in RNN	Merged
#6928	Input Tensors: High Performance Large Datasets via TFRecords	Unmerged
#9965	Change BN layer to use moving mean/var if frozen	Unmerged
#928	Siamese layer	Unmerged

Pull Request Number	Pull Request Title	Status
#442	Caffe models support	Unmerged
#4621	Linear Chain CRF layer and a text chunking example	Unmerged
#3170	Potential New Feature: Hierarchical Softmax	Unmerged
#2183	LSTM with Batch Normalization	Unmerged
#1282	Bidirectional RNN	Unmerged
#368	Caffe support by pranv	Unmerged
#718	3D CNN layers with an example	Unmerged

Chapter 13

Kotlin

By: Jan Gerling, Aurél Bánsági, Kilian Callebaut, Dereck Bridie

Delft University of Technology

Kotlin Banner [1]

13.1　Table of Contents

---

[1] Kotlin Lang, Kotlin homepage, (2019). https://kotlinlang.org/.

13.2   Introduction

Kotlin is a relatively young language, developed by JetBrains, which quickly gained popularity. Announced in 2011 [2], it is a first-class language for Android [3]. Kotlin is used in approximately 5% of the apps available, but in roughly 25% of the "top apps", such as Twitter or TikTok [4].

Kotlin operates primarily in the JVM, is fully interopable with Java, and currently supports Java versions 6 to 9. However, it does target many weaknesses of Java and improves on these. Examples of this are a clear distinction between nullable and non-nullable types and having stronger type inference. These changes lead to developers preferring Kotlin over Java.

Since 2017, Kotlin can also be compiled to JavaScript. This feature was to added to make the language more suitable for all tiers of development [5]. Finally, Kotlin can also be compiled to native code on many different platforms. This is to allow it to run without a virtual machine, which allows Kotlin code to be self-contained. These changes were made to allow the use of Kotlin in as many places as possible, which makes it more appealing to companies.

Since Kotlin is open source, it is possible to contribute to it. Anyone is able to claim issues on YouTrack, the issue tracker developed by JetBrains. The merge process is transparent and visible for anyone. Reviews are done by JetBrains employees, who often requests changed before merging the contribution, if it was deemed suitable. Kotlin is licensed under the Apache 2 license, this means it is not required to sign a Contributor License Agreement to contribute. There is also a `contributing.md` file in the Kotlin repository, which explains the process and which offers a checklist that needs to be completed before submitting a pull rqeuest.

There are different components in the Kotlin project. Of course there is a compiler and parser, which handle the code itself. However, there are also multiple extensions for IntelliJ IDEA, which handles autocompletions or inspections. Most of the *easy* issues on YouTrack focus on the latter. Kotlin is also tested, having over 50,000 thoroughly developed tests. When contributing, one of the requirements is to create some tests for your contribution.

In this chapter, Kotlin will be researched in more detail by first giving an overview of the different stakeholders, before expanding this to a broader context view. This is to properly define the different actors and environment interacting with Kotlin. Then, a more detailed description of the development view is given. This section will contain aspects like standardization and instrumentation. Having looked at the development view, it is then possible to analyze the technical debt of Kotlin. This is important to know since it gives an idea of the current *health* of the Kotlin project. Finally, a new and unique view will be given. This will focus on how developers experience contributing to Kotlin. Over the course of the project, contributions were made to Kotlin. The experiences that occured while making these contributions will be used to enhance this section.

---

[2]Paul Krill, JetBrains readies jvm-based language, (2011). https://www.infoworld.com/article/2622405/jetbrains-readies-jvm-based-language.html.

[3]Maxim Shafirov, Kotlin on android. Now official, (2017). https://blog.jetbrains.com/kotlin/2017/05/kotlin-on-android-now-official/.

[4]AppBrain, Kotlin - android sdk statistics, (2019). https://www.appbrain.com/stats/libraries/details/kotlin/kotlin.

[5]Dmitry Jemerov, Kotlin 1.2 released: Sharing code between platforms, (2017). https://blog.jetbrains.com/kotlin/2017/11/kotlin-1-2-released/.

13.3 Stakeholders

Stakeholders are important to any software architecture, as these people can influence the direction a project moves towards. Therefore, their opinions should be taken into account when evaluating a project as a whole.

The following section gives an overview and analysis of the stakeholders of Kotlin. The stakeholders are categorized based on the stakeholder categories defined by Rozanski and Woods [6].

13.3.1 JetBrains s.r.o.

JetBrains is a software vendor, which develops tools for software developers and project managers. Their motivation to create Kotlin was that they needed a language on the JVM meeting the needs of their company [7]. JetBrains expects that the success of Kotlin will increase sales of IntelliJ IDEA, JetBrains' IDE.

JetBrains role in Kotlin partly comes from manpower: 50+ engineers are working on the project [8], but more roles are fulfilled by JetBrains:

- **Acquirer**
- **Communicator**
- **Developer**
- **Maintainer**
- **Support staff**
- **Supplier**
- **Tester**
- **User**

13.3.2 Kotlin Foundation

The non-profit Kotlin Foundation was created by JetBrains and Google in 2017. The Language Committee adheres to guidelines [9], which describe the decision making process, regarding incompatible changes to Kotlin.

The mission of the Kotlin Foundation is "to protect, promote and advance the development of the Kotlin programming language. The Foundation secures Kotlin's development and distribution as Free Software [. . .]" [10]. Furthermore, the Foundation holds the Kotlin trademark and associated names and logos.

This stakeholder plays a large role as an **Assessor**, overseeing language evolution.

---

[6]N. Rozanski, E. Woods, Software systems architecture: Working with stakeholders using viewpoints and perspectives, Addison-Wesley, 2011.

[7]Dmitry Jemerov, Why jetbrains needs kotlin, (2011). https://blog.jetbrains.com/kotlin/2011/08/why-jetbrains-needs-kotlin/.

[8]JetBrains, Why jetbrains needs kotlin, (2018). https://resources.jetbrains.com/storage/products/kotlinconf2018/slides/KC2018keynote_final.pdf.

[9]Kotlin Foundation, Language committee guidelines, (2019). https://kotlinlang.org/foundation/language-committee-guidelines.html.

[10]Kotlin Foundation, Kotlin foundation, (2019). https://kotlinlang.org/foundation/kotlin-foundation.html#lead-designer.

13.3.3   Lead Language Designer - Andrey Breslav

Andrey Breslav is the President of the Kotlin Foundation, and is a member of both the Board of Directors and the Language Committee. He is also employed by JetBrains as the Lead Language Designer in Kotlin [11]. Andrey is an important stakeholder within Kotlin, being an **Assessor**, **Developer**, **Maintainer**, and **Tester** with high authority.

13.3.4   Google Inc.

Google is the main distributor for Android applications, which can be written in Kotlin. Google has an interest in Kotlin, because the language could empower developers writing applications for their platform to work more efficiently. Google is an **Acquirer** of Kotlin.

13.3.5   Contributors

Contributors are volunteers who are external to JetBrains who contribute to Kotlin as open source developers.

These Contributors can be found in the Kotlin Slack [12], a channel of communication between external developers and Kotlin core members.

These contributors are a part of the **Developer**, **Maintainer**, and **Tester** classes.

13.3.6   Users

Developers writing code in Kotlin directly benefit from improvements to the language. Often, they represent themselves or their development team via public channels such as Twitter or Slack.

Furthermore, people using products written in Kotlin have to be considered, because they have important interests such as application performance, security, and stability.

13.3.7   Suppliers

- *Gradle:* Gradle is a build system that is capable of building Kotlin projects. Their interest in the Kotlin project is twofold: first, their build system should support Kotlin projects, and second, Gradle is also partially written in Kotlin [13].

13.3.8   Communication/Support

Kotlin has various channels to communicate with their community [14].

---

[11]Kotlin Foundation, Kotlin foundation, (2019). https://kotlinlang.org/foundation/kotlin-foundation.html#lead-designer.
[12]JetBrains s.r.o., Kotlinlang slack, (2019). https://kotlinlang.slack.com.
[13]GitHub, Inc., Search .Kt, (2019). https://github.com/gradle/gradle/search?l=Kotlin&q=.kt.
[14]JetBrains s.r.o., Community, (2019). https://kotlinlang.org/community/.

- *Discussion and announcements:* Kotlin Slack, Reddit, Twitter
- *Support:* StackOverflow
- *Gatherings:* User Groups, KotlinConf, Google I/O

13.3.9 Competitors

Kotlin has a multitude of competitors such as Groovy, Scala [15], and Java. All these programming languages are object oriented and focus on a similar group of developers, usually with prior Java experience and offer a comparable feature set.

13.3.10 Enhancers

An important stakeholder type are Enhancers, library developers creating new functionalities for Kotlin language users. Therefore, these stakeholders add value to Kotlin by creating their own solutions within the scope of Kotlin.

13.3.11 Tools

An additional class of stakeholders create tools for Kotlin developers, to increase their productivity and ease software development. Their interests are to generate income, add value to Kotlin and further spread Kotlin.

- *detekt:* Static code analysis [16]
- *dokka:* Documentation engine for Kotlin [17]
- *IntelliJ IDEA*

13.3.12 Power/Interest grid

A power/interest grid shows the power of a stakeholder to influence a project and their interest in the project [18].

13.3.13 Influence on Stakeholders

An onion diagram shows the interaction of stakeholders with a project and effect of the project on the stakeholders [19] [20]. The diagram consists of five layers:

- **Core** - the project goal
- **Layer 1** - closely involved stakeholders

---

[15]Eugen Paraschiv, The state of java in 2018, (2018). https://www.baeldung.com/java-in-2018.

[16]Artur Bosch, Detekt, (2019). https://arturbosch.github.io/detekt/.

[17]Ilya Ryzhenkov, et. al., Documentation engine for kotlin, (2019). https://github.com/Kotlin/dokka.

[18]G. Johnson, K. Scholes, Exploring corporate strategy, prentice hall, Europe. (1999).

[19]Business Analyst Learnings, Stakeholder onion diagram: A practical guide, (2019). https://businessanalystlearnings.com/ba-techniques/2013/1/22/how-to-draw-a-stakeholder-onion-diagram.

[20]BAwiki, Stakeholder onion diagram, (2019). http://bawiki.com/wiki/techniques/stakeholder-onion-diagram/.
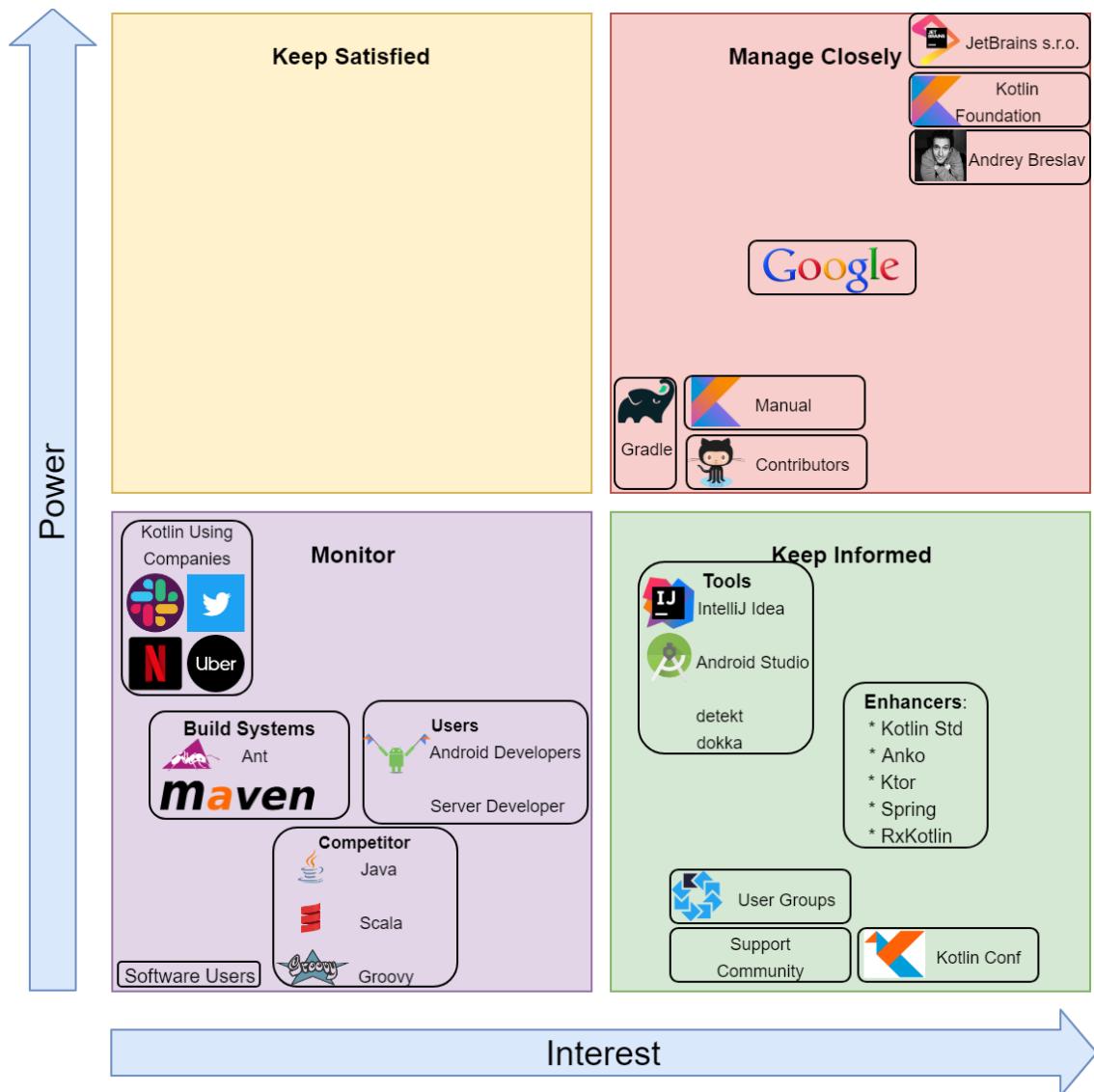
Figure 13.1: Power/Interest grid with the stakeholders of the Kotlin project.

- **Layer 2** - highly affected stakeholders
- **Layer 3** - Beneficiaries
- **Layer 4** - wider environment

13.4 Context View

The context view describes the relationships, dependencies and interactions between the system and its environment. This is visualized in the diagram. Each element of the diagram will be described in more detail in this section, along with an overview of the scope of the system.

13.5 System Scope

Kotlin is a object-oriented programming language aiming to "provide a more concise, more productive, safer alternative to Java that's suitable in all contexts where Java is used today" [21]. Kotlin tries to eliminate some of Java's biggest issues like nullity and excessive coding [22]. In this regard, Kotlin's scope is to provide an gradual upgrade from Java, by integrating Kotlin code into their existing Java Project. In order to enable developers to quickly adopt Kotlin, workflow and feature set are very similar.

13.5.1 Context Model

At the heart of the context model is the language itself. Kotlin is written in Java and Kotlin, and runs on the JVM by default. It can also be compiled to JavaScript [23] and to native code [24]. Kotlin natively makes use Java or JavaScript functions, when the correct compiler configuration is selected

13.5.1.1 Developers

Kotlin is an open source project to which anyone can contribute on GitHub. These contributions are reviewed by JetBrains employees and merged into the project. Together with Google, JetBrains sponsors Kotlin. They are both represented in the Kotlin Foundation. The Language Committee is a part of the Kotlin Foundation and reviews breaking changes. Andrey Breslav is the lead Language Designer of the language. These people make all decisions regarding the development of the project and evolution of the language [25], as well as overviewing the community contributions.

13.5.1.2 Related Processes

GitHub is used for contributing and project management of Kotlin. JetBrains uses their own software YouTrack for discussing and tracking issues, features, and bugs. TeamCity is used for continuous integration

---

[21] D. Jemerov, S. Isakova, Kotlin in action, Manning Publications Company, 2017.

[22] Android Developers, Develop android apps with kotlin, (2019). https://developer.android.com/kotlin#safer-code.

[23] Kotlin Foundation, Kotlin javascript overview, (2019). https://kotlinlang.org/docs/reference/js-overview.html.

[24] Kotlin Foundation, Kotlin/native for native, (2019). https://kotlinlang.org/docs/reference/native-overview.html.

[25] Kotlin Foundation, Kotlin foundation, (2019). https://kotlinlang.org/foundation/kotlin-foundation.html#lead-designer.
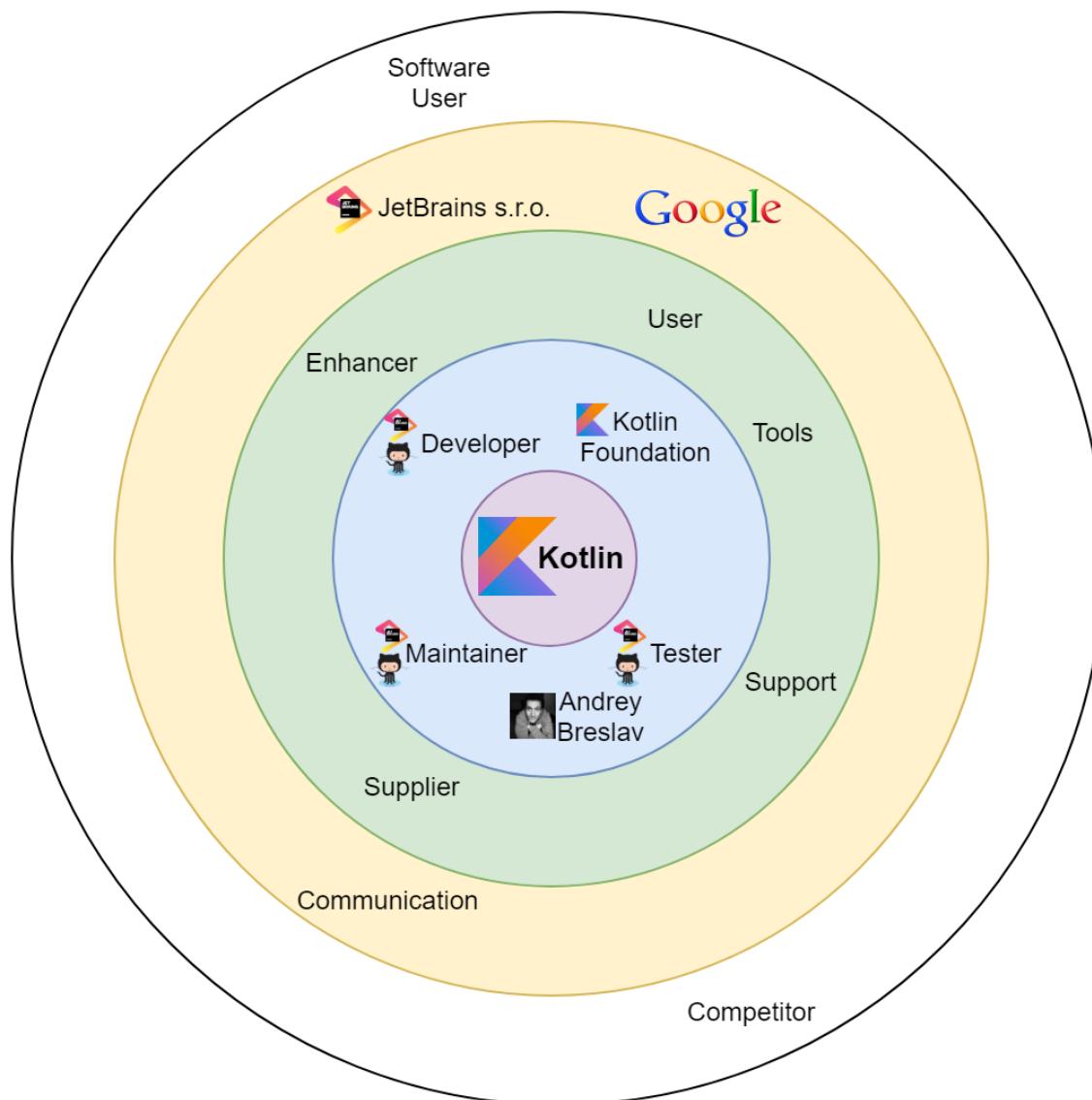
Figure 13.2: Onion diagram of the key stakeholders of the Kotlin project and stakeholder types
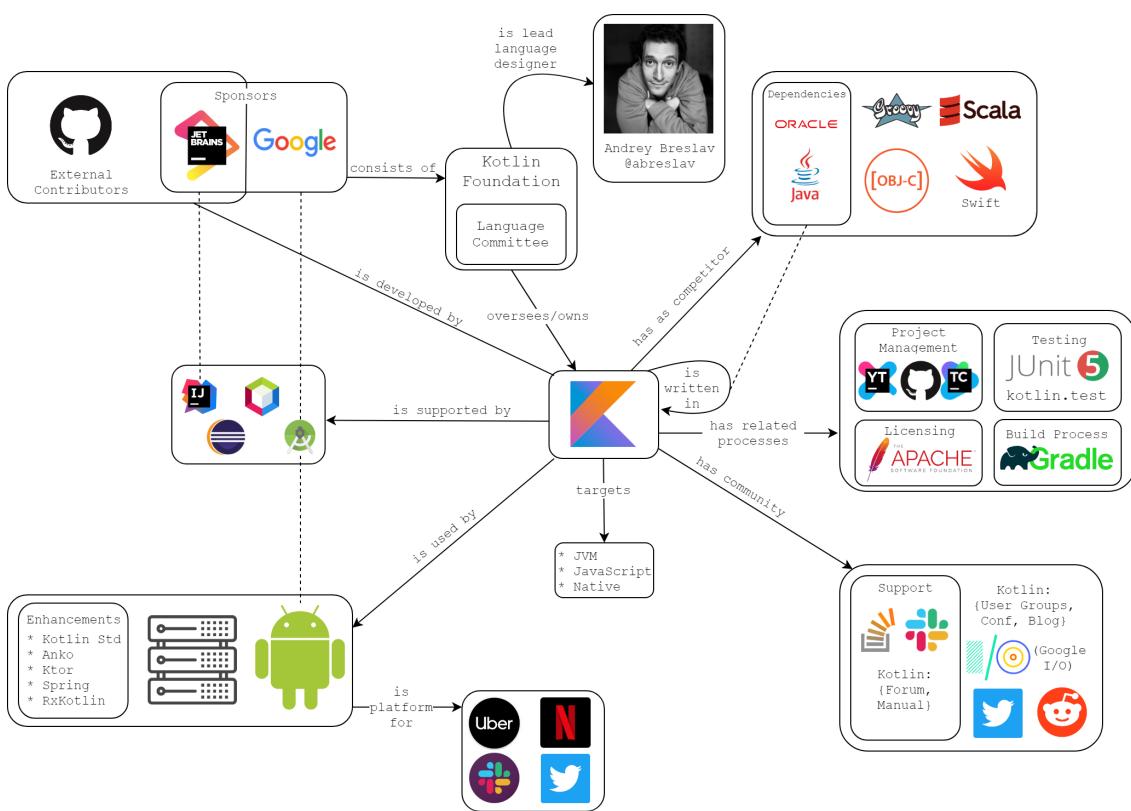
Figure 13.3: Context View of Kotlin

and deployment. Finally, other related processes are: JUnit for testing (with Kotlin extensions), Apache 2 License [26] and Gradle for building.

13.5.1.3 Community

The Kotlin community has multiple channels to communicate their feedback for Kotlin. Conferences like Kotlin Conf serve to present updates from the Kotlin development team, as well as user's talks about the state and usage of the language. The official Twitter [27] and the Kotlin subreddit [28] provide channels to post news updates, pose questions and discuss issues with the language where the JetBrains' Kotlin team frequently responds. There are also more direct channels used to support developers like StackOverflow and Slack.

13.5.1.4 Supported by

The Kotlin language is supported by many IDEs, such as JetBrains' own IDE IntelliJ IDEA, Google's Android Studio, and third party IDEs like NetBeans and Eclipse.

13.5.1.5 Used by

A big advantage of using Kotlin is the support from Google for its Android platform. Another major use cases is in server side code. Examples of popular applications that have been developed using Kotlin are Uber [29], Netflix, Slack, and Twitter [30]. For Kotlin development there are enhancing frameworks available.

13.6   Pull Request Analysis

In this section we analyze the top ten most commented pull requests for Kotlin on GitHub [31], that were merged and rejected. We want to identify project integrators, main reasons why pull requests are rejected and which changes could lead to successful merge requests. Our strategy to identify these patterns, was to find the core of the discussion in PRs and discover reoccurring issues.

13.6.1   Most commented merged PRs

Most PRs in this list seem to have a high comment number not due to controversy, but because of people misunderstanding or needing a more thorough explanation what the reviewer requested to change. This leads to a longer conversation in which the reviewer looks at the changes multiple times. A particularity of the Kotlin repository is that most of the pull requests are merged manually.

---

[26] JetBrains s.r.o., License, (2019). https://github.com/JetBrains/kotlin/tree/master/license.
[27] Kotlin Twitter, Kotlin twitter, (2019). https://kotlinlang.org/foundation/kotlin-foundation.html#lead-designer.
[28] Community, Kotlin reddit, (2019). www.reddit.com/r/kotlin.
[29] Kotlin Lang, Kotlin homepage, (2019). https://kotlinlang.org/.
[30] Android Developers, Develop android apps with kotlin, (2019). https://developer.android.com/kotlin#safer-code.
[31] JetBrains s.r.o., The kotlin programming language, (2019). https://github.com/JetBrains/kotlin.

13.6.2 Most commented unmerged PRs

Unmerged and closed PRs usually seem to be declined due to inactivity or because the changes are not what is deemed relevant for Kotlin at the times. At times, a senior developer for Kotlin at JetBrains, Mikhail Glukhikh, uses the PR system to perform extensive code reviews, which leads to a large amount of comments.

13.6.3 Integrators

Kotlin uses a pull-based development model for doing reviews. Developers review the code and usually review the code within a few days. The reviews appear to be quite thorough, but most pull requests are merged either without changes required, or after one round of review. Most of these pull requests tag an open issue for Kotlin, which makes them easy to accept. However, as stated before, the changes are often manually merged. Most *failing* pull requests are not merged due to the changes not falling in line with the direction of where Kotlin is going.

Pull requests are integrated by senior developers at JetBrains, who review the code and request changes. One person, Mikhail Glukhikh [32], is assigned very often to review PRs.

13.6.4 Contact with JetBrains

We tried to establish direct communication with the JetBrains Kotlin Team. Unfortunately, this did not succeed, thus, all communication, e.g. asking question was done on the Kotlin Slack [33].

13.7 Technical Debt

In this section, technical debt in Kotlin is identified and analyzed, by dividing it into four areas: code debt, test debt, design debt, and communication about debt.

Code debt is important since it shows how maintainable the code is. Test debt is an important factor since lacking tests and bad test habits can have a large impact on the project and are hard to resolve in future. Design debt describes the debt caused by the module structure of the project, and dependencies on other code. Finally, communication about debt shows that the developers are aware of the debt.

13.7.1 Code Debt

In order to properly analyze Kotlin, automatic analyzation tools are necessary. For this section we used detekt [34], a framework made specifically for Kotlin, and the Statistic plugin [35] for IntelliJ IDEA. Statistic allows to check several relatively simple metrics, e.g. logical lines of code (LLoC). Line based metrics have limited validity, but can provide basic insights.

---

[32]GitHub inc., Mikhail glukhikh, (2019). https://github.com/mglukhikh?
[33]JetBrains s.r.o., Kotlinlang slack, (2019). https://kotlinlang.slack.com.
[34]Artur Bosch, Detekt, (2019). https://arturbosch.github.io/detekt/.
[35]Tomas Topinka, Statistic, (2019). https://plugins.jetbrains.com/plugin/4509-statistic.

Detekt is more powerful, but only analyses Kotlin files. It provides strong metrics, such as McCabe cyclometic complexity (MCC). Additionally, it also estimates the time to fix the code debt and points out specific problems.

13.7.1.1 Results
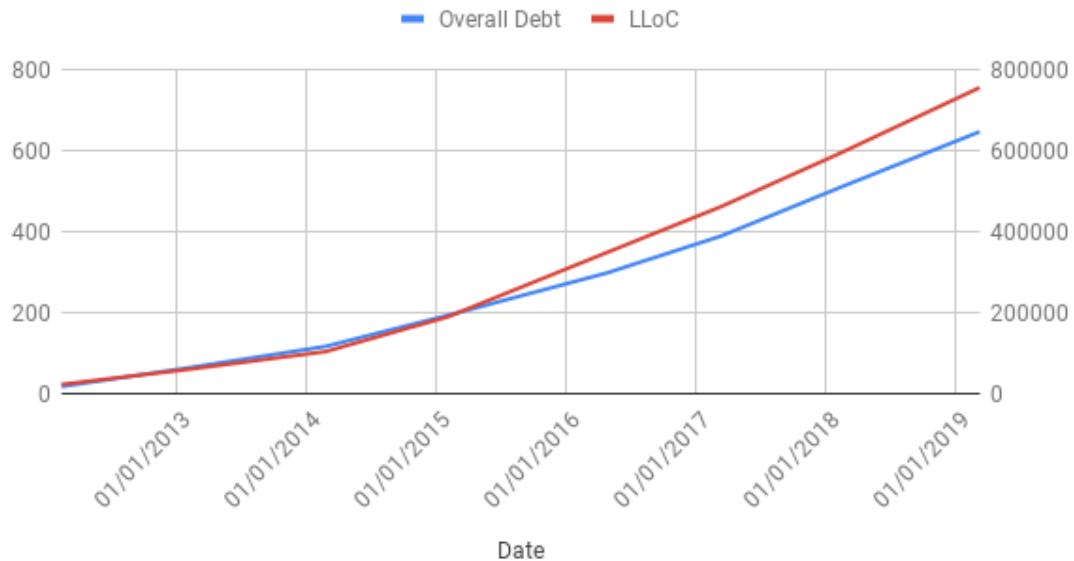
It is evident that there is some serious code debt in Kotlin, negatively impacting maintenance due to a lack of clarity and explanations for the methods [36]. This creates a growing threshold for contributors in terms of time wasted on understanding the system.

Statistic showed that roughly 12% of the content of Java and Kotlin files are comments. When accounting for the licensing in every file, around 10% lines of Code (LoC) are comments. Inspecting large files shows that most of the classes do not contain many comments. As this appears to be consistent across the entire project, this is an indicator for large code debt.

Detekt shows that the MCC and code smells per 1000 LLoC are high for a project of its size. Detekt estimates that the time it would take to fix all code debt in Kotlin at 642 days, although no reasoning is given for this estimate. However, some of these code smells might not be issues; for instance, Detekt does not exclude generated files.
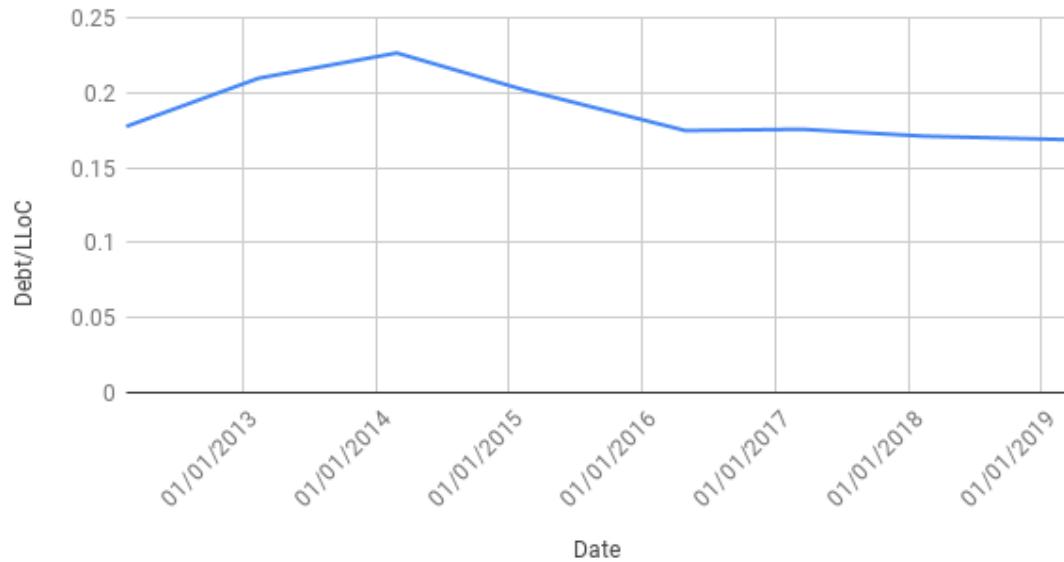
By running Detekt on several commits, once every year, code debt evolution was analyzed. The first figure below shows that while the overall debt has grown massively, it has decreased compared to the LLoC. The second figure below shows that the relative number of code smells has slightly decreased over time.

---

[36]E. Allman, Managing technical debt., Commun. ACM. 55 (2012) 50–55.

Code Smells/LLoC over time

13.7.1.2 Open TODOs

Unresolved TODOs in code are a good metric for known technical debt in the project, the analysis of the TODOs history can be seen in the figure below.

In 2016, the amount of TODOs grew significantly from about 850 to over 3000. The developers added many TODOs in testing, especially in `idea` and `compiler`. Likely due to Kotlin being released around this time they did not resolve these issues right away, increasing the testing debt further.

Furthermore, the overall number of open TODOs is increasing, reaching more than 4000 in March 2019, indicating significant technical debt in the `idea` and `compiler` components. Not all this is necessarily bad, as long as they can be repaid [37]. The increasing number of TODOs does not however bode well as this also means the time to resolve these TODOs keeps increasing.

13.7.2 Test Debt

In this section the testing debt of Kotlin is analyzed, by looking at bugs and analyzing the test quality.

---

[37]E. Allman, Managing technical debt., Commun. ACM. 55 (2012) 50–55.

TODOs history with major components
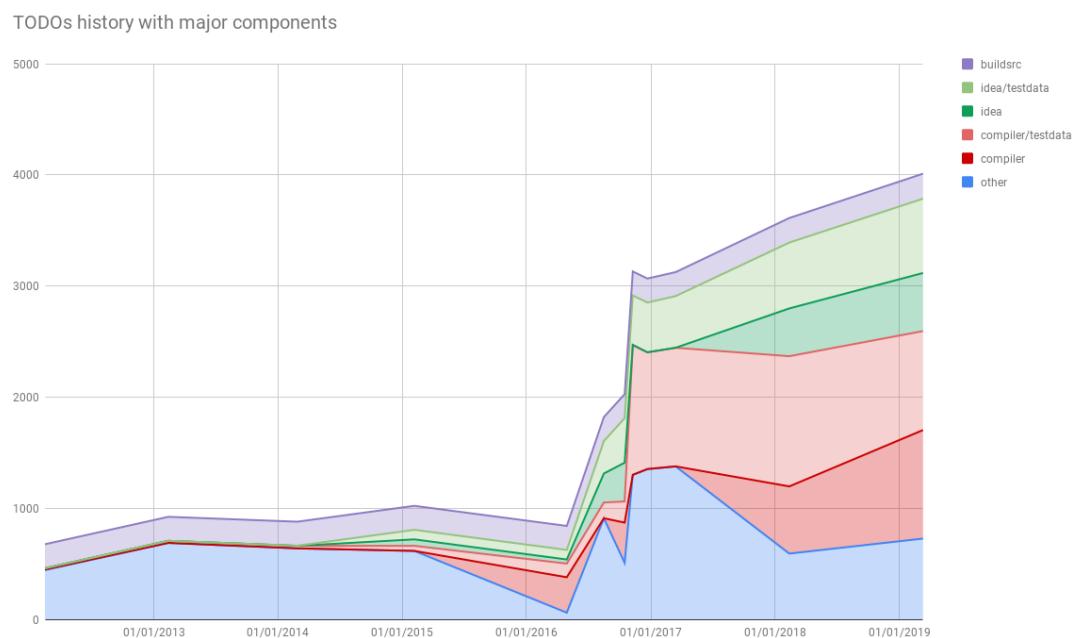
Figure 13.4: History of TODOs since 2012 with main components, stacked amount of TODOs

13.7.2.1   Bugs

On YouTrack [38], all bugs in Kotlin are tracked with priorities and affected subsystems. The analysis focused on major and critical bugs, because these are the ones with the highest impact.

In the period from 2011 until March 2019, 2428 major and 629 critical bugs were fixed in Kotlin. However, 774 major and 5 critical bugs remain unfixed. An analysis of the issue tracker shows that critical bugs are likely to be resolved within a few months. However, major bugs stay unfixed much longer, some even for multiple years (see examples 1 [39], 2 [40], and 3 [41]). The main components affected by bugs are IDE, Frontend, Tools, and Backend (see the figure below).
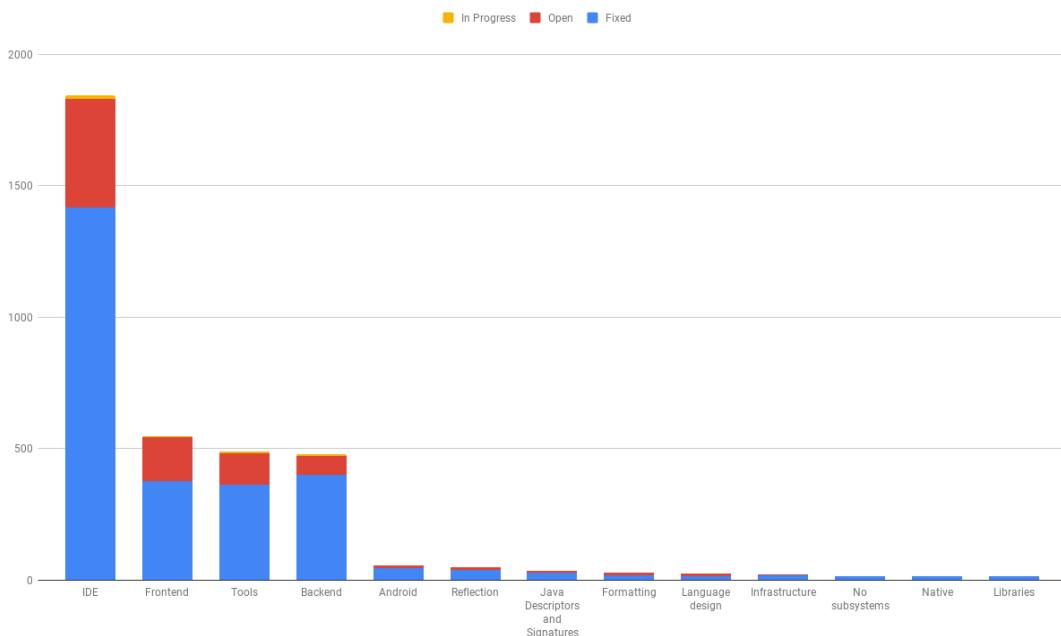
Figure 13.5: Top 10 affected subsystem by bugs with stacked amount of bugs and current state (March 2019)

13.7.2.2   Test Quality

The current state of testing in Kotlin is difficult to assess, because analyzing test coverage is very difficult due to the project having more than 55.000 tests, while lacking test coverage tools.

TeamCity and IntelliJ IDEA do not support code coverage for the Kotlin language [42]. Since JetBrains likely does not track test coverage, they have a presumably high potential of test debt. Furthermore, the

---

[38]JetBrains s.r.o., Kotlin (kt) - bug and issue tracker, (2019). https://youtrack.jetbrains.com/issues/KT.

[39]Svetlana Isakova, KT-9070, (2015). https://youtrack.jetbrains.com/issue/KT-9070.

[40]Hannes D., KT-11978, (2016). https://youtrack.jetbrains.com/issue/KT-11978.

[41]Eugene Petrenko, KT-16213, (2017). https://youtrack.jetbrains.com/issue/KT-16213.

[42]Irina Megorskaya, Code coverage, (2015). https://confluence.jetbrains.com/display/TCD9/Code+Coverage.

high amount of TODOs and their slow decline suggests JetBrains is aware of their testing debt, but is not prioritizing it.

To improve testing standards (see section Standardization of testing) JetBrains could create quality requirements and guidelines for open source developers. Furthermore, test reviews should be mandatory for pull requests, because testing is not checked in pull request reviews (see section Pull Request Analysis).

13.7.3 Design Debt

Since Kotlin is a large project, the internal dependencies are difficult to evaluate (see in section Module Organization). It is evident that the internal structure is quite complex with many intertwined modules. This is an indicator of high coupling, and thus large code debt. Meaning it is tough to add changes to the system.

While analyzing the dependencies of Kotlin, older version of libraries were found. For instance, it appears that JUnit 4 is still used, which has been out-of-date for over a year.

One of the major dependencies of Kotlin is Java. Java 6 was deprecated in december 2018, but is still a listed dependency of Kotlin. This is probably due to Android still heavily depending on Java 6. Since Android is a target platform for Kotlin, this might be why Kotlin still wants to support Java 6.

13.7.4 Communication about Debt

In this section we discuss how Kotlin developers communicate and discuss about technical debt.

Filtering issues on YouTrack on the keywords "refactoring", "refactor", "maintain", "maintenance", and "debt", does not return any discussions about technical debt. Furthermore, there are no labels for technical debt of any kind. Filtering the commit messages with the same keywords results in more than 2000 commits with refactorings, but very few discussions about maintenance or refactoring. Additionally, only 20 TODOs (see the figure above) contain the words "refactor" or "maintain". On their Slack [43] there are no dedicated channels for refactorings, testing or maintenance.

All these findings indicate that Kotlin developers very rarely discuss technical debt.

13.8 Development View

To analyse the development of Kotlin, we will take a look at the architecturally significant concerns. The approach as specified by Rozanski and Woods [44] will be used in this section.

---

[43]JetBrains s.r.o., Kotlinlang slack, (2019). https://kotlinlang.slack.com.

[44]N. Rozanski, E. Woods, Software systems architecture: Working with stakeholders using viewpoints and perspectives, Addison-Wesley, 2011.

13.8.1 Concerns

13.8.1.1 Modules

The Kotlin project repository is organized into modules and orchestrated using Gradle. Analyzing the build files of the project showed there are 199 modules.

A dependency graph has been created to show the links between each subproject. From this graph, it can be seen that sub-projects tend to have many dependencies to other sub-projects. Kotlin has multiple third-party dependencies such as, Java, JUnit, ANTLR 4, and Google Guava.

The top 10 nodes by amount of times referenced are the following:

Name	Times Referenced	Description
`:kotlin-stdlib`	68	Contains the Kotlin standard library [@stdlib]. Used in all Kotlin projects.
`:idea`	63	Contains all editor integration between IntelliJ IDEA and Kotlin adjacent projects.
`:compiler:frontend`	59	Contains classes and methods to analyze Kotlin code, resolve types, and show diagnostic compiler messages.
`:compiler:util`	52	Contains various utilities ranging from constants to class conversions.
`:compiler:tests-common`	44	Contains all kinds of boilerplate for creating language tests. Used in many projects.
`:compiler`	42	Groups all sub-modules together: contains no functionality on its own.
`:compiler:frontend.java`	39	Contains behavior when the Kotlin compiler interacts with a Java source file.
`:compiler:cli`	32	Contains code for command line tools `kotlinc`, `kotlin-dce-js`, etc.
`:core:descriptors`	27	Contains classes for naming Kotlin language elements and types.
`:idea:core`	27	Contains all editor integration between IntelliJ IDEA and Kotlin.

Table 301 - Top 10 nodes by amount of times referenced

This list of top referenced nodes also shows points of common processing: nodes that are included often must contain some kind of functionality that is useful to share and distribute across other modules.

Furthermore, there have been 7 general layers of modules identified.

The module structure graph shows the general relationship between module layers.
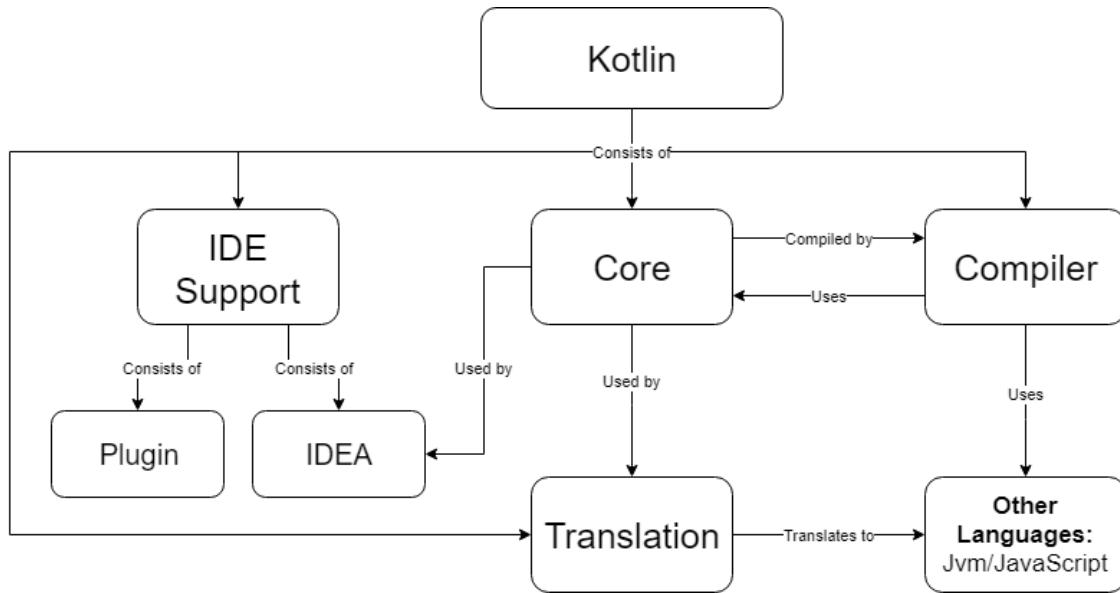
Figure 13.6: Diagram of the module structure.

13.8.1.2   Standardization of design

There are not a lot of explicit rules or guidelines for standardizing the design of the Kotlin source code. The contributer's guide [45] only specifies rules in regards to creating the pull request and communicating what this contributes to the project. The only code related check that needs to be done is that tests have to be written, run, and passed for the contribution. Aside from that, the current workflow appears to be to first look at similar contributions, and then attempt to be consistent with their style.

There is, however, a coding conventions document [46] for the Kotlin language. Since the Kotlin project is mainly written in Kotlin, this means these guidelines still apply for the source code contributions. Any pattern that can be used to make the contributed code more efficient would be mentioned in the review, but they are not defined beforehand.

13.8.1.3   Standardization of testing

As was mentioned before, there is a requirement to create tests when submitting a pull request. Since there is a standardised way to create tests, this is a relatively easy process. For each type of feature, there is a package in which tests can be added. Tests can then be generated by Gradle, which can then also execute tests.

According to the contributor's guide for Kotlin [47] every developer has to implement, run and pass all tests for their contributions. There are no further requirements or guidelines for testing.

---

[45] Nikolay Krasko, Contributing, (n.d.). https://github.com/JetBrains/kotlin/blob/master/docs/contributing.md.

[46] Kotlin Lang, Contributing, (2019). https://kotlinlang.org/docs/reference/coding-conventions.html.

[47] Nikolay Krasko, Contributing, (n.d.). https://github.com/JetBrains/kotlin/blob/master/docs/contributing.md.

JUnit 4 and automatically generated test collections are used to simplify test creation. For most types of test there is a file in the tests package that automatically generates code. This code will test the files it will find in the corresponding package in testdata. The tests are generated by running the "generate all tests" configuration in IntelliJ IDEA. All these tests are stored in multiple test suites regarding different modules of the system, e.g. testIDE or compiler.

13.8.1.4   Instrumentation

For a programming language, instrumentation is a key property. Kotlin supports code tracing, debugging, profiling, and data logging.

To enable developers to debug their software, Kotlin complies [48] to the DWARF 2 debugging data format standard [49], allowing developers to use breakpoints, stepping, and inspections of types and variables.

Currently (March 2019), both Kotlin and IntelliJ IDEA do not have a native profiler for Kotlin, but decompiling Kotlin code to Java code allows to use JVM Profiler [50] shipped in the JVM. The compiler can target JVM, native, and JavaScript, which means tools for those environments can be used.

Every Throwable [51] object, the base class for exceptions and errors, contains a stacktrace array [52] with the last frames before occurrence. Like most languages, exceptions contain a message, stack trace and, if applicable, a cause [53].

Kotlin provides some options to profile software. Compiler Instrumentation [54] can be setup and used to profile an application during compilation. A more specific tool is the measureTime [55] function, measuring the runtime of a certain code block.

Logging can be done in a similar way to logging in Java. Furthermore, in IntelliJ IDEA many issues are identified and shown to the developer during code writing. This is an feature of Kotlin, giving direct feedback to Kotlin developers.

13.8.1.5   Codeline Organisation

The codeline organisation [56] refers to the way the directory structure is managed and tested, while being regulated via configuration management. The management of the building and tests is done using Gradle. The testing itself is done using JUnit 3 and 4.

Furthermore, JetBrains uses TeamCity for an continuous integration and deployment process [57]. TeamCity

---

[48] Kotlin Lang, Debugging, (2019). https://kotlinlang.org/docs/reference/native/debugging.html.

[49] Unix International, DWARF debugging information format, (1993). http://dwarfstd.org/doc/dwarf-2.0.0.pdf.

[50] Zlata Kalyuzhnaya, IntelliJ idea 2018.3 eap: Git submodules, jvm profiler (macOS and linux) and more, (2018). https://blog.jetbrains.com/idea/tag/jvm-profiler/.

[51] Kotlin Lang, Throwable, (2019). https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-throwable/index.html.

[52] Kotlin Lang, StackTrace, (2019). https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/stack-trace.html.

[53] Alexander Udalov, Andrey Breslav,, Exceptions, (2019). https://kotlinlang.org/docs/reference/exceptions.html.

[54] Alexander Udalov, Andrey Breslav,, Exceptions, (2019). https://kotlinlang.org/docs/reference/exceptions.html.

[55] Kotlin Lang, MeasureTimeMillis, (2019). https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.system/measure-time-millis.html.

[56] N. Rozanski, E. Woods, Software systems architecture: Working with stakeholders using viewpoints and perspectives, Addison-Wesley, 2011.

[57] Kotlin TeamCity, Kotlin teamcity, (2019). https://teamcity.jetbrains.com/viewType.html?buildTypeId=Kotlin_dev_Compiler&branch_Kotlin_dev=%3Cdefault%3E&tab=buildTypeStatusDiv.

runs a multitude of build agents on the latest master, creating new builds roughly every 3 hours, if all tests pass.

The file structure itself is confusing for newcomers. The project has not been structured in a clear and logical form, nor is there any documentation on how the structure for the Kotlin project should be organized. Nevertheless, an attempt was made to comprehensibly identify all the packages of the project and their purpose, which can be found in the appendix.
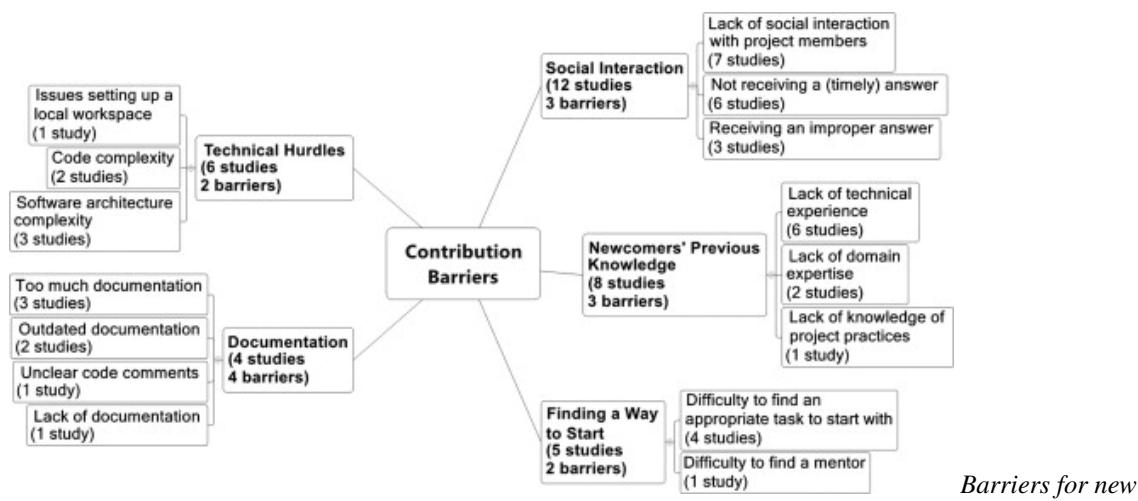
For people that are wanting to contribute, it is the easiest to start with looking at the `idea` folder, since this contains plugins for IntelliJ IDEA, such as inspections. Most of the up-for-grabs issues are also centered around this package.

The main issue with this set-up is, that there is no clear indication or grouping of any functionality. In order to contribute to Kotlin, a developer has to analyze other commits and pull request to get a basic understanding of the project.

In conclusion, there are issues with documentation and clarity in the Kotlin project. It would be beneficial to add more logical structuring and documentation, which would lead to people having an easier time contributing.

13.9   Contributors Perspective

Kotlin is a project that partially relies on open source contributions in order to evolve and has welcome contributors since it was made open source in 2012 [58]. In order for these contributors to work effectively, they should be supported throughout the process. In this section the journey of contributiors throughout making contributions and proposing new features will be explored, before giving recommendations for potential improvements. This journey was derived from three personas (see Appendix Personas) and the contribution barriers (see the figure below) identified by Steinmacher et al [59].

Barriers for new

---

[58] Ann Oreshnikova, Kotlin goes open source!, (2012). https://blog.jetbrains.com/kotlin/2012/02/kotlin-goes-open-source-2/.

[59] I. Steinmacher, M.A.G. Silva, M.A. Gerosa, D.F. Redmiles, A systematic literature review on the barriers faced by newcomers to open source software projects, Information and Software Technology. 59 (2015) 67–85.

open source contributors found in a literature review by Steinmacher et al. in 2015 [60].

13.9.1   Orientation

In the orientation phase, the contributor sets up the development environment, picks an issue and gets familiar to the proceedings of Kotlin.

Finding all necessary files and documents on the Kotlin repository [61] such as `Readme.md` and `contributing.md` [62] was no problem for the contributors.

Furthermore, understanding how to propose a new feature for Kotlin via the Kotlin Evolution and Enhancement Process (KEEP) [63] is easy for new developers as it is well described. All things considered, the Kotlin KEEP process can take several month to years and requires long term effort.

Unfortunately, the issue tracker is confusing for contributors, because issues on the issue tracker that are up-for-grabs [64] is not as simple as suggested. Many of these filtered issues are already claimed, because contributors can claim issues by leaving a comment or already have a pull request pending, which is either tagged as such or not [65]. Thus, finding an issue to grab requires lots of manual inspection of the issue tracker and the Kotlin repository [66].

The build instructions are clear: JDKs 1.6, 1.7, 1.8, and 9 are required, with the former two being soft requirements [67]. It is not clear for developers if they can use the OpenJDK [68] instead of the Oracle JDK [69], as it appears to be operable as well.

To improve the start for contributors, the issue tracker could allow developers to claim an issue formally by adding a tag, such as "Grabbed" or "Doing". Then, applying a filter would give a clear overview of the issue state. Furthermore, the Pull Request tag should be properly maintained such that pull requests that are pending do not show up in the filtered list. Additionally, it should be clear whether or not the OpenJDK can be used to contribute to Kotlin, as it is easier to get and install in certain environments.

Because KEEP is can be slow and very time consuming JetBrains should add further information on how to request features for Kotlin without making a large commitment.

13.9.2   Development and Testing

In this section, the development and testing experience for an open source contributor for Kotlin after an issue was chosen will be explored. This involves understanding the project organization, documentation,

---

[60] I. Steinmacher, M.A.G. Silva, M.A. Gerosa, D.F. Redmiles, A systematic literature review on the barriers faced by newcomers to open source software projects, Information and Software Technology. 59 (2015) 67–85.

[61] JetBrains s.r.o., The kotlin programming language, (2019). https://github.com/JetBrains/kotlin.

[62] Nikolay Krasko, Contributing, (n.d.). https://github.com/JetBrains/kotlin/blob/master/docs/contributing.md.

[63] JetBrains s.r.o., KEEP - kotlin evolution and enhancement process, (2019). https://github.com/Kotlin/KEEP.

[64] JetBrains s.r.o., Tag: Up For Grabs and state: Open, (2019). https://youtrack.jetbrains.com/issues/KT?q=tag:%20%7BUp%20For%20Grabs%7D%20and%20State:%20Open.

[65] Burak Eregar, Convert java to kotlin dialog ask yes/no question but shows ok/cancel buttons, (2018). https://youtrack.jetbrains.com/issue/KT-27869.

[66] JetBrains s.r.o., The kotlin programming language, (2019). https://github.com/JetBrains/kotlin.

[67] JetBrains s.r.o., The kotlin programming language, (2019). https://github.com/JetBrains/kotlin.

[68] Oracle Corporation, OPEN jdk, (2019). https://openjdk.java.net/.

[69] Oracle Corporation, Java se downloads, (2019). https://www.oracle.com/technetwork/java/javase/downloads/index.html.

requirements, and implementation of the selected issue.

Starting to contribute to Kotlin can be difficult for developers, because the project structure is unclear and very difficult to assess (see section Module Organization). Additionally, many functions especially used for inspections are uncommented and can be difficult to understand for a new contributor.

Since the 23th of March 2019, documentation especially for inspections and quickfixes has been added [70], teaching new contributors how to start with intentions and in which directories to look. This is a major improvement on the user experience, as a majority of Kotlin's easy issues are related to Inspections and Intentions [71].

Writing tests is a challenge: the contribution guidelines [72] are unclear about the requirements for these tests, stating that related tests must be run locally and passed, and that all inspections should have automatic tests [73], but no mention of approaches or other requirements exist. The testing habits of other developers are also differing widely from very little testing [74] to sound tests [75]. Furthermore, many of the tests contain various test metadata, such as 'Problem: NONE', e.g. `inapplicable1.kt` [76], but they are not listed or documented.

Furthermore, there is no method to run tests with coverage: the related button in IntelliJ IDEA is broken for this project, leading to an error instead of coverage data.

The Kotlin developer community on their Slack [77] and helping with many issues developers face.

Documentation should be added for components that a new contributor is likely to use. When working on a new project, code documentation is an important part of understanding how components interact with each other. An example of this is `ProblemsHolder.registerProblem` [78], a function all inspections should call. However, there are three method overloads; documentation could inform a user which overload is appropriate for which situation. This is partially done with the new documentation for inspections and quickfixes [79].

Coverage information should be fixed for the Kotlin project so that information like statement coverage and branch coverage can be seen. This helps contributors identify missing test cases and improves the quality of contributor tests, positively impacting test debt (see section 4 - Testing Debt).

To help new contributors, a mentorship program could be created, where more experienced open source developers would assist newcomers. This approach is recommended by the open source guide as well [80].

[70]Nikolay Krasko, Intention/quickfix/inspection quick notes, (2019). https://github.com/JetBrains/kotlin/blob/master/docs/intentions_inspections_quickfixes.md.

[71]JetBrains s.r.o., Up for grabsIDE. Inspections and Intentions, (2019). https://youtrack.jetbrains.com/issues?q=%23%7BUp%20For%20Grabs%7D%20%23%7BIDE.%20Inspections%20and%20Intentions%7D.

[72]Nikolay Krasko, Contributing, (n.d.). https://github.com/JetBrains/kotlin/blob/master/docs/contributing.md.

[73]Nikolay Krasko, Intention/quickfix/inspection quick notes, (2019). https://github.com/JetBrains/kotlin/blob/master/docs/intentions_inspections_quickfixes.md.

[74]Mikhail Glukhikh, Kotlin master, (2018). https://github.com/JetBrains/kotlin/tree/master/idea/testData/inspectionsLocal/redundantSuspend.

[75]Toshiaki Kameyama, Mikhail Glukhikh, Kotlin master, (2019). https://github.com/JetBrains/kotlin/tree/master/idea/testData/inspectionsLocal/moveLambdaOutsideParentheses.

[76]Toshiaki Kameyama, Kotlin master - inapplicable1.kt, (2018). https://github.com/JetBrains/kotlin/blob/master/idea/testData/inspectionsLocal/moveLambdaOutsideParentheses/inapplicable1.kt.

[77]JetBrains s.r.o., Kotlinlang slack, (2019). https://kotlinlang.slack.com.

[78]Anna Kozlova et al., Intellij-community master, (2018). https://github.com/JetBrains/intellij-community/blob/4999f5293e4307870020f1d0d672a3d35a52f22d/platform/analysis-api/src/com/intellij/codeInspection/ProblemsHolder.java.

[79]Nikolay Krasko, Intention/quickfix/inspection quick notes, (2019). https://github.com/JetBrains/kotlin/blob/master/docs/intentions_inspections_quickfixes.md.

[80]Sophie Shepherd et al., Best practices for maintainers - learning to say no, (2016). https://opensource.guide/best-practices/#learning-

Mentors can help newcomers to understand the technical details, coding guidelines, and rules of the project and thus, increase their productivity and quality of contributions.

13.9.3   Submission

This section addresses the final steps for a contribution: submitting a pull request to the Kotlin repository and the review by project maintainers.

Submitting pull requests works without problems. In our experience the time between code submission and first review can vary between a day and three weeks. After it is accepted, the change is integrated into the `master` branch and is integrated in the project, automatically picked up by TeamCity to be included in a following release (see section 3 - Development Process).

The timespan of the PR review process should be lowered. This barrier was also identified by six of the reviewed studies by Steinmacher et al. (see the diagram of the module structure above) and is also addressed in the opensource guide [81]. A long period of waiting can be demotivating for new contributors, because it can make your contribution feel insignificant after a lot of effort was poured into it. A good response time recommended in literature is below 48 hours [82].

13.10   Conclusion

In this chapter, Kotlin was analyzed. Being a fairly new, but popular, language, Kotlin has attracted a lot of attention. Thus, it was important to first get an overview of the different stakeholders. It was found that evolution of the language is controlled by the Kotlin Foundation, and more specifically the Language Committee. Almost all of the people in this foundation represent one of the other two major stakeholders, JetBrains and Google.

All stakeholders are part of a broader context view. In this context view, it was found that Android is the largest platform for Kotlin. It also identified competitors such as Java or Swift. The tools that are used to develop JetBrains were also found, such as YouTrack for issue tracking, and TeamCity for continuous integration. Another important discovery is that Kotlin can be compiled to work on the JVM, JavaScript, and to native code.

Because Kotlin is open-source, anyone can contribute to the project. This means that there are more people able to look at and contribute to the project, leading to a potential decrease in technical debt. Looking at merged and unmerged pull-requests, discovered was that most PRs are usually merged to master after some time has elapsed.

Having identified the context of Kotlin, the next step was to zoom in to the development view. In this view, the different modules were identified and a diagram was made. Furthermore, the standardization of both design and testing was looked at. It was found that there is not really a standard for design, but there is a

---

to-say-no).

[81]Sophie Shepherd et al., Best practices for maintainers - documenting your processes, (2016). https://opensource.guide/best-practices/#documenting-your-processes.

[82]C. Jensen, S. King, V. Kuechler, Joining free/open source software communities: An analysis of newbies' first interactions on project mailing lists, in: 2011 44th Hawaii International Conference on System Sciences, 2011: pp. 1–10. doi:10.1109/HICSS.2011.264.

standard way to write tests, since there is a testing framework included in the project. There is also plenty of instrumentation available, helping developers to create and debug their code.

While researching the development view, time was also spent analyzing the potential technical debt of Kotlin, split in four different categories. It was found that Kotlin has significant debt in all categories. It was found that it would take an approximate 650 days to fix all the code debt, containing issues such as a lack of comments and a comparatively cyclomatic complexity. Additionally, there is a large number of open TODOs and bugs, however, these numbers are slowly declining over the years. This fact, coupled with no apparent analysis for test coverage, shows a sign of high test debt. Since there is also no open conversations that are visible about technical debt

The final perspective is the viewpoint of developers contributing to Kotlin. The section identified potential barriers for contributors based on experience and research. Three personas were created, through which the contribution process was analyzed and recommendations were given to improve this process. The most important recommendations were: improve documentation and test coverage, decrease the time waiting for review, and simplify the process of claiming issues.

Summarizing the results, Kotlin is a language that is starting to mature, and the fact that is open-source is helping with this. However, there are still a lot of issues that needs to be fixed, especially with regards of how easy it is for a new developer to contribute.

13.11 Appendix

Directory	Purpose
.idea	IDE support
android-studio	IDE support
annotations	Compiler
ant	Building
build-common	Building
compiler	Compiler
core	Standard library
custom-dependencies	Building
docs	Documentation
eval4j	Java Interpreter
Generators	Building
gradle	Building
idea	Functionalities and tests
idea-runner	running
include	compiling
j2k	Conversion Java to Kotlin
jps-plugin	Compiling Java
js	Compiling Javascript
Konan	Compiling Native Code
Libraries	Functionality
license	Apache licensing
plugins	Functionality

Directory	Purpose
prepare	Building
resources	Building
spec-docs	Documentation
test-instrumenter	Test compiling
third-party	Compiler
ultimate	IDE Support

Chapter 14

Lila (lichess.org)

14.1 Introduction

Lila is the open-source code behind the libre, no-ad chess website lichess.org. The name is a playful combination of Li(chess in Sca)la, given the backend code is written in Scala. The philosophy of the project is nicely summed up in the project's maintainer's github profile description: "Benevolent dictator of lichess.org, a hippie communist chess server for drug fueled atheists." [1]. The site is a major player in the online chess world with more than one million games played every day [2].

14.2 Stakeholder Analysis

As the lichess project is open-source with a rather small group of core contributors some of the stakeholder types identified by Rozanski and Woods are not fully applicable [3]. Specifically we don't have aquirers since lichess is open and free. We identified some new roles not covered in the book and changed some roles slightly (see the stakeholder subsection).

14.2.1 Github activity

Our stakeholder and decision making analysis is based to a great extent on the github project. Specifically we selected the most discussed pull requests, 10 that were rejected, 10 that were merged (a list can be found in the appendix). We also looked at contributions in general over the lifetime of the project.

---

[1]"Ornicar github." [Online]. Available: https://github.com/ornicar. [Accessed: 18-Mar-2019].

[2]"About lichess." [Online]. Available: https://lichess.org/about. [Accessed: 01-Apr-2019].

[3]N. Rozanski and E. Woods, Software systems architecture: Working with stakeholders using viewpoints and perspectives. Addison-Wesley, 2011.

14.2.2 Stakeholders

There are three main **developers**, of which the number of commits are shown above. Ornicar is by far the most active [4]. He has started Lichess and remains the main developer. Over time, several other developers have helped him. Originally, Clarkerubber and Unihedro were his main cocontributers, but since 2015 the only other constantly active developers were Niklasf and Isaacl [5],[6].

Developer	Commits total	Commits 2018
Ornicar	25922	2146
Niklasf	935	309
Isaacl	478	71

There is a group of **moderators** including more than twenty users on the website who help moderate the forum and enforce rules. They perform various moderating tasks such as forum management and the banning of cheating or misbehaving players. There are only 9 people who open issues and while forum names and github names don't always correspond, it seems that the moderators are also the ones posting issues on github, which means they are also **testers** of the system. As the forum is the main communication channel moderators are all **communicators** and as it is the place for users to ask questions, they also fulfill the **support** role.

Ornicar seems to be the person responsible for deployment and adminstration of the servers, thus serving as **production engineer** and **system administrator**. He is also the main **integrator** and his roles as production engineer and system administrator shine through in this role, as he often comments on performance when merging pull requests and even rejects pull requests if the performance is lacking [7]. The other integrator is Niklasf. Most often both comment on a pull request before it is merged. While Isaacl regularly contributes to the discussion, he has to ask Ornicar or Niklasf to merge.

Lichess has a yearly running cost of $102,792.3, which is entirely covered by donations from **sponsors**. This money mostly comes from monthly donations from Patreon supporters and from a webshop. The Patreon supporters do not have any special status within Lichess, but Lichess is very much dependent on them. The only commercial parties that Lichess seems to have relations to are the various servers where Lichess is hosted, making them the only **suppliers**. The way the hosts are talked about in the "thank you" section on the website seems to imply that Ornicar is on good terms with the hosts and possibly enjoys significant discounts. This means they might be a combination of sponsor and supplier.

Most of the **users** are from Germany, Canada, USA, Russia and the UK [8]. These users don't have to pay or even watch ads to play on the website as it is completely free. What Lichess do get from their users are donations and developers [9], which it heavily depends upon. Secondly, next to the casual gamers you have people who like to improve their skills using Lichess. These people can get their games analysed and can even get a coach via Lichess. Thus, the website is used by all sorts of users of different skill levels. The users of all different levels are therefore crucial for the existence of Lichess. Almost all other stakeholders are also part of this large, diverse group. For example, the **sponsors** are all users, the **moderators** are

[4]"Ornicar github." [Online]. Available: https://github.com/ornicar. [Accessed: 18-Mar-2019].

[5]"Niklasf github." [Online]. Available: https://github.com/niklasf. [Accessed: 18-Mar-2019].

[6]"Isaacl github." [Online]. Available: https://github.com/isaacl. [Accessed: 18-Mar-2019].

[7]"Upgrade to reactivemongo 0.12." [Online]. Available: https://github.com/ ornicar/lila/pull/2115. [Accessed: 18-Mar-2019].

[8]"Alexa site info lichess." [Online]. Available: https://www.alexa.com/siteinfo/ lichess.org. [Accessed: 18-Mar-2019].

[9]"Lichess donations." [Online]. Available: lichess.org/patreon. [Accessed: 18-Mar-2019].

recruited amongst the users, experienced users **support** each other through the forum and they also function as **communicators**.

We estimate the different degrees of power and interests different stakeholders have in the following diagram.

Power

High

Ornicar

Niklasf Isaacl

Suppliers

Moderators

Interest

Low

High

Users

Sponsor (patreon)

Testers

Coach

High profile
players

Low
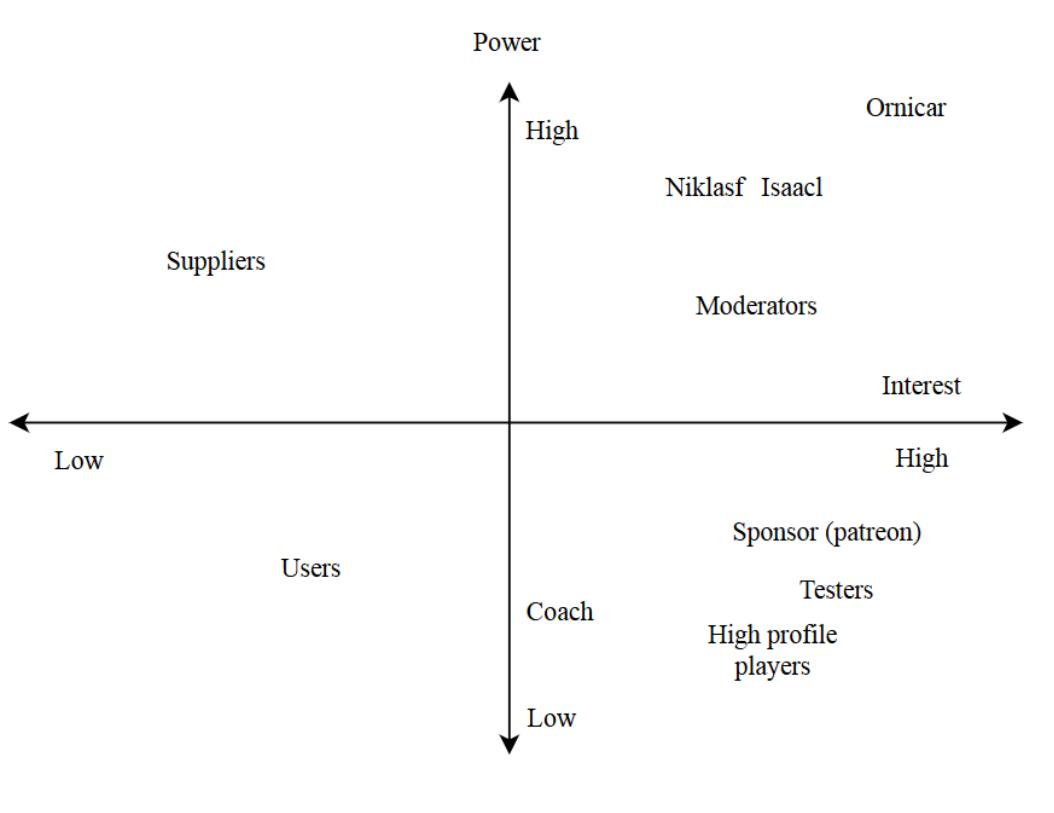
Figure 14.1: Power Interest Grid

14.2.3 Decision making process of merging pull requests

The Lila project does not have written guidelines for merging pull requests. Most declined pull requests (that is, a pull request that did not get merged),were lacking in quality, completeness or generally not needed (e.g [10],[11]).

Interviewing some active developers (revoof and Toadofsky) on Lila's discord channel gives us some insight in what developers and mainteners look for:

- "Ultimately it's up to the maintainers to merge Pull Requests."

---

[10]"Fix for #4538." [Online]. Available: https://github.com/ornicar/lila/pull/ 4591. [Accessed: 18-Mar-2019].
[11]"Add 100 quotes (3)." [Online]. Available: https://github.com/ornicar/lila/ pull/1472. [Accessed: 18-Mar-2019].

- "The things I look for during reviewing or merging are: coding style, things it could potentially break and verify that it doesn't break these things and as far as features are concerned I ask myself the question if the feature is worth the complexity and cost of maintenance down the line."

When asked if the project could benefit from having a set of guidelines and a process that is well documented. The developers react rather lukewarm:

- "Probably not, to be honest. That sounds very rigid. I feel it's more important to review code and give feedback in a timely manner".
- "Lichess is probably not the best project for developers who are just starting out"

14.3   Context View
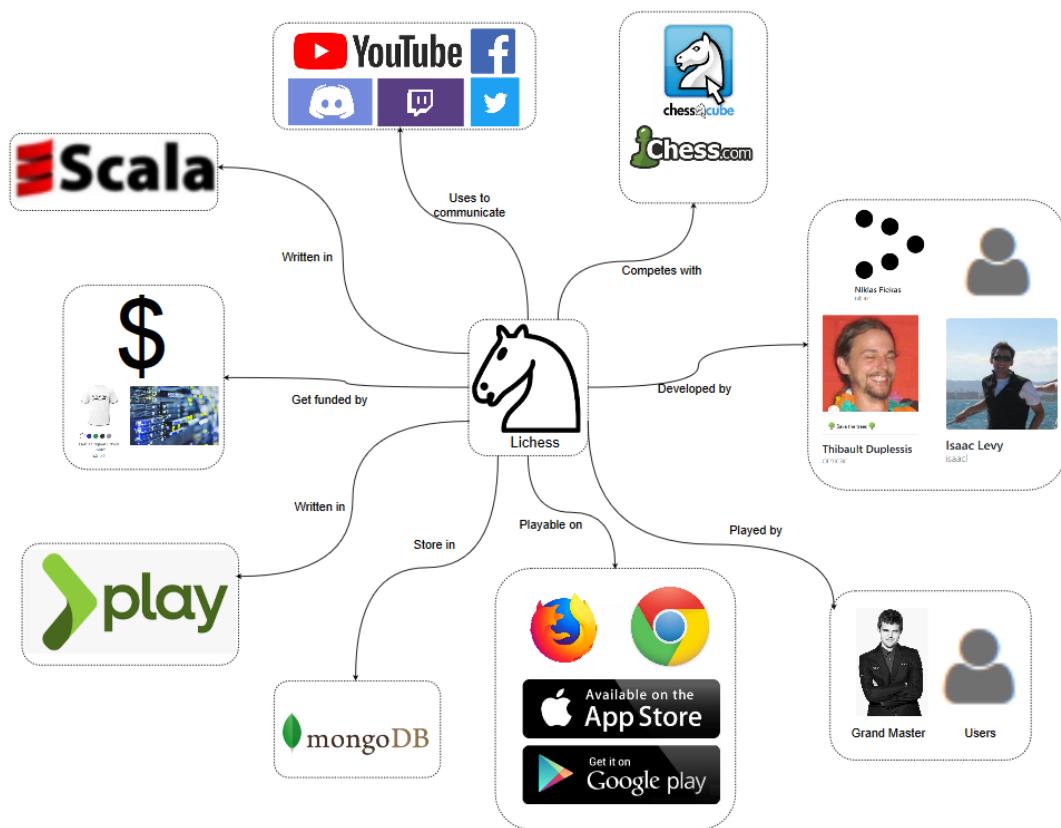
Figure 14.2: Context View

14.3.1 Responsibilities

Lichess primarily offers a platform for people to play online chess. It has extensive matchmaking and ranking of players, as well as offering many different play modes (including learning games). This website is not only for everyday users, also Carl Magnussen amongst other grand masters use lichess to play chess online [12].

Lichess has mobile apps (iOS, Android) and a web version. The browsers that are supported are the desktop versions of Mozilla Firefox and Google Chrome. On Opera, Safari and Internet Explorer or Edge you are able to play as well, however these are unsupported and can therefore have performance issues.

14.3.2 External Entities

In order to communicate with their users and provide content Lichess uses different platforms. They use Twitter, Facebook, Twitch, Discord, Youtube and they even have a forum on their own website. As mentioned in the stakeholder view, Lichess gets donations and sells merchandise to keep the website running. When the users want to report bugs are feature request, they can do this in the forum or directly on Github. Lichess competes with other platforms such as chess.com or chess cube, which differ slighly through more paid content.

14.4 Development View

Due to space constraints for the following analysis of the development view and technical debt we focus on the backend code of the project, which is written in Scala 2.11.

14.4.1 Dependencies

The Play Framework 2.4 is used to provide the typical web application backend infrastructure and MongoDB is used for persistance. Other major dependencies are:

- scalachess (game logic, also developed by ornicar)
- akka actors (concurrent message passing)
- kamon (monitoring)

14.4.2 General architecture

The backend logic is structured in 69 modules, totalling around 59k lines of code. Server-side rendering is used to serve the web pages to the users, the Play framework has its own templating engine twirl for this purpose. The frontend executables are developed in a separate subproject, which are copied to the appropriate resource folders in the build process. A mix of typescript and javascript is used without a big frontend framework. An overview of this very high-level architecture is depicted in the following figure.

---

[12]"Magnus carlsen wins the first lichess titled arena." [Online]. Avail- able: https://lichess.org/blog/WjRTPScAAJXo7r5s/magnus-carlsen-wins-the- first-lichess-titled-arena. [Accessed: 18-Mar-2019].
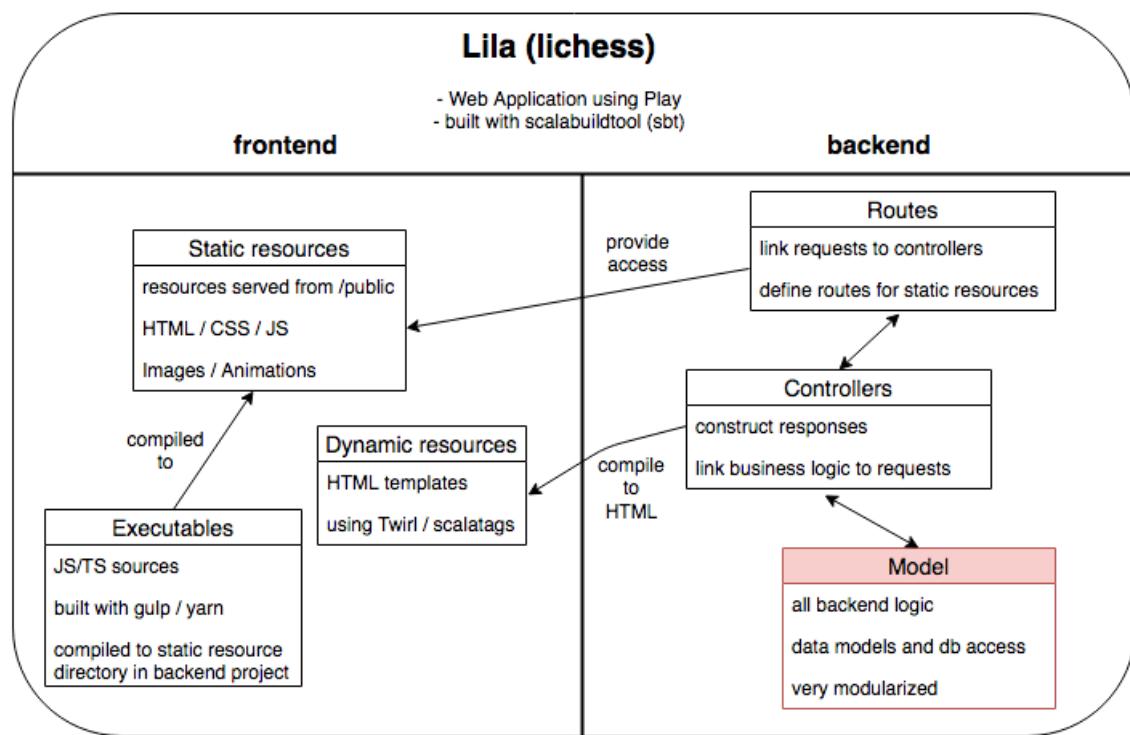
Figure 14.3: Project Anatomy

The Play framework provides the infrastructure for routing and handling of HTTP requests including error responses in a typical MVC (Model-View-Controller) project layout. Lila follows the standard structure, defining all routes in a route configuration file, controllers to handle requests and the business logic in many separate modules. Views are server-side rendered by the use of twirl templates. Overall it seems like the project is not tied to the Play framework alot, but the heavy use of twirl templates would make a switch hard. A plan to migrate to another more lightweight template libary scalatags is planned, but only very few parts have been migrated.

In the following we focus just on the Model which is the heart of any project, here all the business logic specific to Lila resides.

14.4.3   Modularization

With 69 modules the logic is very modularized and separated roughly according to functionality. There are 59 000 lines of code in total, distributed amongst packages as seen in the following figure:
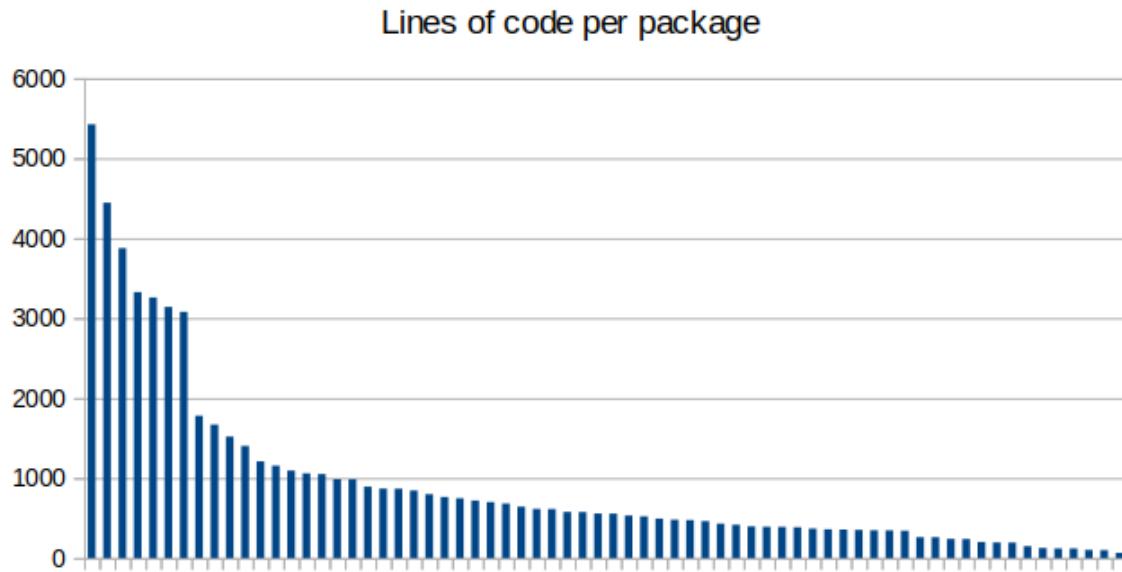
Figure 14.4: Code distribution

The first six modules break the curve with significantly more lines of code than the others. These bigger and presumably more important modules are listed below:

- tournament (47 files, 5.4k loc)
- study (38 files, 4.4k loc)
- game (45 files, 3.9k loc)
- common (52 files, 3.3k loc)
- security (36 files, 3.3k loc)
- user (31 files, 3.1k loc)

Mostly these correspond to the major features offered by lichess, with the common and security module offering more general services. In the module dependency figure below we can see how there are quite a few general purpose modules that are depended on throughout the project's codebase. Our diagram depicts the general direction of dependencies from the general purpose modules within lila at the top to the external dependencies and utility modules at the bottom. The modules that have arrows going to the bottom of their block depend on modules in the block directly below it or further down.
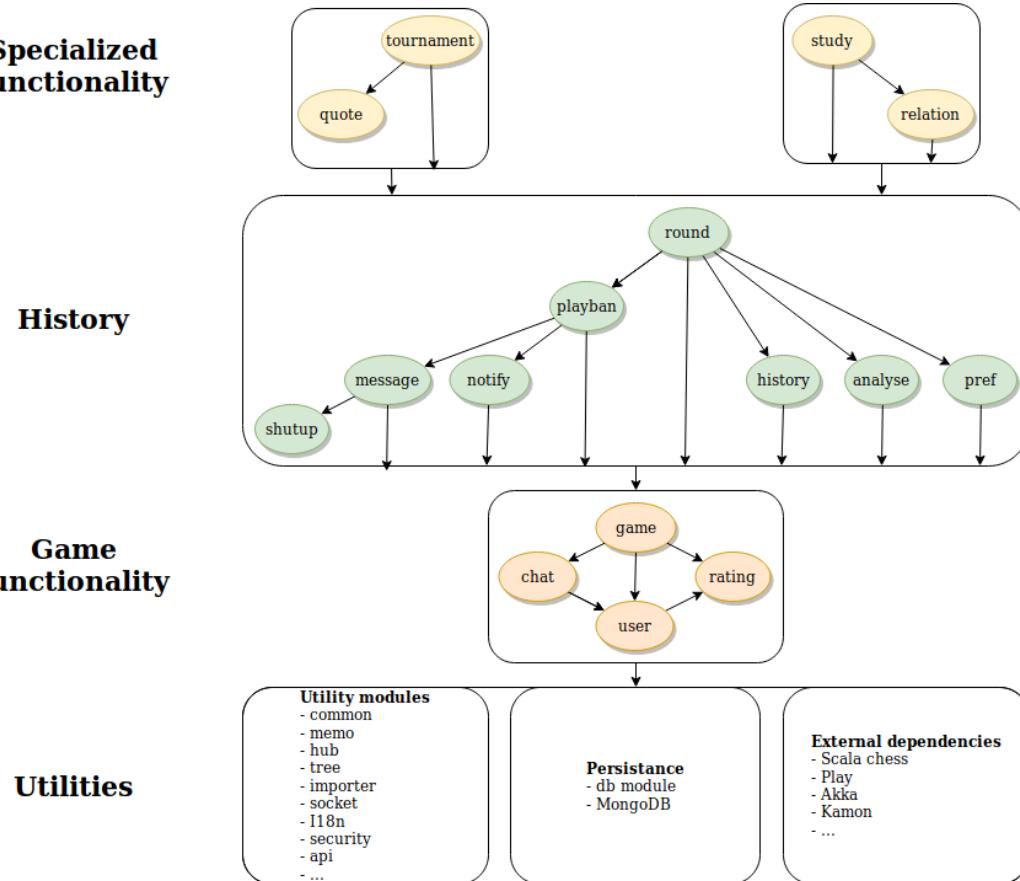
Figure 14.5: Module dependencies

At the core of the codebase is the game module which is depended on by much of the codebase. It offers the logic needed for the playing of a chess game, the major functionality of the lichess website (for this it heavily depends on the scalachess library). All troughout the codebase the utility modules such as common, security and api are used. A critical component is found in the db module which centralizes database access functionality using the reactive mongo library.

Higher-level functionality such as messaging, notifications and analysis per chess game are organized around the round module. This layer facilitates working with mutliple games of chess. It includes history and analysis modules, as well as playban, which checks whether a player is allowed to play the next round. This could still be considered part of the core as it provides all the functionality needed that revolves around the

playing of a single chess game. On top of that, there are the study and tournament layers. Both depend heavily on the layers beneath them and the ability to play, record and analyse multiple games. Overall the project is modularized along its core features and makes for quite a natural splitting in components. This makes development easier as functionality is directly linked to the internal codebase structure.

14.4.4 Common Patterns and Utilities

The project makes use of a few special versions of common data structures such as futures, sequences, primitives and trees which are defined in amongst others the common module, the tree module and the coordinate module. These are essentially wrapper classes or slightly tweaked versions of the types found in the Scala standard library. Furthermore to facilitate concurrent messaging the akka library is used for its event bus offering interfaces to subscribe to events and for its actor-based message passing. There are also some lila modules that handle concurrency, such as socket, hub and memo. The i18n module offers internationalization utilities for translating UI elements to the client's language. In addition, security is handled in a separate module.

14.4.5 Development and Releases

The lila project has a wiki page on how to setup a development environment and a small contribution guide. They mostly help with having the project running locally and include tips on what kind of feature requests are welcomed. However there is not much information on how and where to contribute to lila. The project uses scalariform to enforce a certain code style, but not many rules are used. Mostly rules from the general Scala Style Guide are inherited. Furthermore a few conventions, termed "Lilaisms", are defined. These include some helper functions and wrappers around common functional programming structures such as options and futures, which are heavily used throughout the codebase.

In general the contribution process starts with either an issue reported on github or a feature request that is discussed in the lila forum and then posted as an issue. Some tagging of issues helps categorize issues into feature requests and bugs. Ui issues are also tagged separately as "no scala" issues.

It is unclear what is the exact process of releasing new versions and deploying to production in the lila project. It is not documented and asking ornicar whether there is a continous integration pipeline he replied: "there's no such thing, I deploy from local machine". We previously also assumed that no continious integration analysis is done due to the lack of tests and no mention of this anywhere.

14.5 Technical Debt

In order to gain insight into the technical debt the project has accumulated over time we use SonarQube and Codacy. For SonarQube we use both the default sonar plugin and the sonar-scala plugin. The multi-module configuration required us to make a separate script to generate a sonar configuration, as it would be quite tedious to configure all 67 modules. The separate modules are relatively small, making the codebase score very high on maintainability. Though, the technical debt of the individual modules differ significantly. In total, 287 code smells are found based on the sonar rules and 2287 issues are found based on the codacy rules.

14.5.1   Issues

Based on our sonar analysis we observe a total technical debt of 6 days based on issues alone, with most issues belonging to the code smell category. Inspecting the technical debt we find that the codebase is littered with commented out code, unreachable code, code duplication and unused code. Specifically the duplicate use of string literals makes the code less maintainable and violates the DRY princicple.

We observe that some sources are very difficult to read as they have a very high cognitive load, with the WMMatching.scala file in the commons module scoring a Cognitive Complexity of 265. Another issue potentially decreasing maintainability are very large objects containing more than 30 methods (we found 31 such objects and classes). Complexity is also an issue with the amount of method parameters which exceed 8 for a total of around 20 methods. Readability is further decreased by heavy use of unnamed tuples in subexpressions (110 such usages were detected).

Furthermore, we notice that the project contains quite a few warnings regarding security. Notably that no input sanitizing takes place and the usage of in-secure cookies. Most importantly, the RegExes being used to verify URLs make an easy target for DDoS attacks (due to the usage of wildcard characters).

We also notice that some modules make poor use of Scala's functionalities, relying on mutable fields or solving problems in a procedural manner as opposed to a functional approach. This is not necessarily technical debt, but it is non-idiomatic use of the language which decreases cohesion of the project as a whole. Some modules have significantly more technical debt than other modules, something that could have been prevented if the Lila repository made use of CI tools such as sonar. Below a table can be seen with the modules that contain most code smells:

Module	Count
quotes	48
db	40
common	13
game	12
tournament	10
study	8
insight	7

It is hard to quantify the amount of technical debt within the lila project, however the big contrast in terms of code smells between the different modules shows the project is not in a very consistent state. A lot of work would be required to reduce these inconsistencies by refactoring and more cohesive language usage.

We assume that one of the biggest causes of the built up technical debt is due to the absence of structured code reviews, continious integration and other tools that may help with reducing issues with code. Therefore we also suggest that the lila team takes setting up a CI/CD pipeline into strong consideration.

It should be noted, however, that due to the fact the code is spread accross multiple modules, most code can be easily modified as the modules are not tightly coupled with one other. In other words: lila has a very maintainable codebase. Which probably has helped keep the project as popular as it is today, with an ever increasing user count.

As mentioned previously the focus of our analysis is on the Scala portion of the codebase, however the project also contains a small amount of Java and JavaScript code containing a few issues as well. The

majority of issues (>90%) stem from the Scala code though, so we can assume the majority of the technical debt is within the backend logic.

14.5.1.1 Diving deeper into the code smell

Upon manually inspecting all 287 code smells we decided whether a code smell is a "true" code smell or a false positive. See a table below of our results:

	Count
False Positives	143
True Positives	144

Surprisingly, many code smells (~ 50%) are a false positive. Examples of this are code smells such as code used for localization of the application (translation into different languages). Though, one must argue whether storing language translations inside a scala file is a best practice as this also flagged UTF-8 warnings as some other alphabets are also included in this file which was flagged as a code smell.

Most true positives we found are mis-use of Scala, such as using variables and procedural code use. Furthermore, we think that commented out code does not belong in a production environment. Also, some true code duplication can be found in the project where it is clear code is copy-pasted.

Interestingly, we found code duplication across modules, which Sonar did not seem to flag as code duplication. We assume that if this would be possible for Sonar to detect we might have found more "true" code smells in the "code duplication" category.

Overall, upon manual inspection of the issues flagged by Sonar and Codacy, we think that the project is in a healthy state given the low amount of code smells that are actually significant to the project. As to why this is the case, without structured code reviews and automated code analysis, is probably the fact that this project has solely been developed by one (very experienced) Scala developer since 2013, only seeing significant contributions from other users around 2016.

14.5.1.2 Resolution

As we would like to help Lila give insight into their project, we proposed to set-up sonar analysis for the project. After some discussion with the maintainer of the project they were happy to have Sonar in place to help scan the project on merge requests.

However, after setting up the sonar analysis we did notice that the time required for performing the Travis build had more than doubled. Builds that normally took 9 minutes now took over 20 minutes. Adding to this, the maintainer, ornicar, stated the following:

- "I fail to see the value."

- "https://sonarcloud.io/project/issues?id=lila&resolved=false&types=CODE_SMELL is all about functions starting with an upper case letter. These are standard in Play projects, including lila, and I don't want to change that."

- "https://sonarcloud.io/project/issues?id=lila&resolved=false&types=BUG is all about the .scala.html templates, which are in the process of being entirely replaced with .scala scalatags."

- "This PR is intrusive, mainly because it requires a new sbt plugin, and I don't see the benefits for lila."

Even though we did not manage to have sonar integrated into lila, we did try our best to help the project lila gain more insight into code quality. Moreover, we did contribute to lila through other means, solving bugs and implementing features.

14.5.1.3   Historical analysis

Due to the complex structure in which the project is set up, it is difficult to perform a quantitive analysis on the history of the project as we would have to re-generate the sonar properties files for each of the 30,000 commits. Even choosing a set of commits and manually setting up sonar, codacy and code coverage is a complex process of changing build tools, property files and compiler errors. Because of this we perform qualitative analysis and inspect the timeline of the project manually.

When browsing through the GitHub repository however, we do note that the project has almost solely been developed by one person and the multi-module structure has remained intact ever since the project started. Therefore we assume that this project has slowly accumulated technical debt (especially tests).

14.5.2   Testing Debt

The Testing Framework Specs2 is used for writing unit tests. However, overall the project's backend code is tested very little, the code has a whopping 2.68% branch coverage (the scoverage sbt plugin was used to obtain the coverage data). Only 16 out of 68 modules actually have unit tests. Only the following 7 modules have >5% coverage:

- lila.game
- lila.base (part of common)
- lila.importer
- lila.analyse
- lila.blog
- lila.shutup
- lila.socket

Among these only the game and base package are the most central modules to the codebase. Other big modules contributing core functionality are not tested at all.

Due to the very low amount of test coverage we estimate that the amount of time required to get to a respectable (at least 70%) test coverage is substantially large that it probably is not feasible writing tests for the remaining 50000 lines of code that have not been covered. Overall for the evaluation of technical debt within the lila project that lack of tests is a major issue with the potential to cause unmaintanability along the road. Given that it is the second largest chess website currently in use, the absence of test is quite worrying.

Ornicar, the maintainer of the project, states that he does not think functional tests are of importance to lila after being asked why there are little tests. ## Information View

The lila project is by its nature as a chess application with extensive study and analysis tools an information-heavy project, that requires quite some bookkeeping. As mentioned before MongoDB is used as a database for persisting information. It is a NoSQL document-store database which saves structured information in a format similar to JSON. Advantages of this kind of storage over traditional relational databases include an easier mapping of data structures to the supported storage format and better locality when retrieving related components. It is also easier to evolve the data and its relations over time, which comes with a penalty of no guaranteed schema (unlike SQL databases).

The lila project stores its different data in various collections (a collection is the document store equivalent to a table in a relational database), such as:

- Games
- Users
- Tournaments
- Study units

In the following we will describe the most important components of lila's data model and how and when information is persisted. Furthermore we end with a short discussion on data ownership and privacy aspects of the project.

14.5.3 Static Data Model

Central to the data persisted is the user, which is associated with various components such as rankings, teams, games etc. A high-level overview of these relationships can be seen in the following diagram.
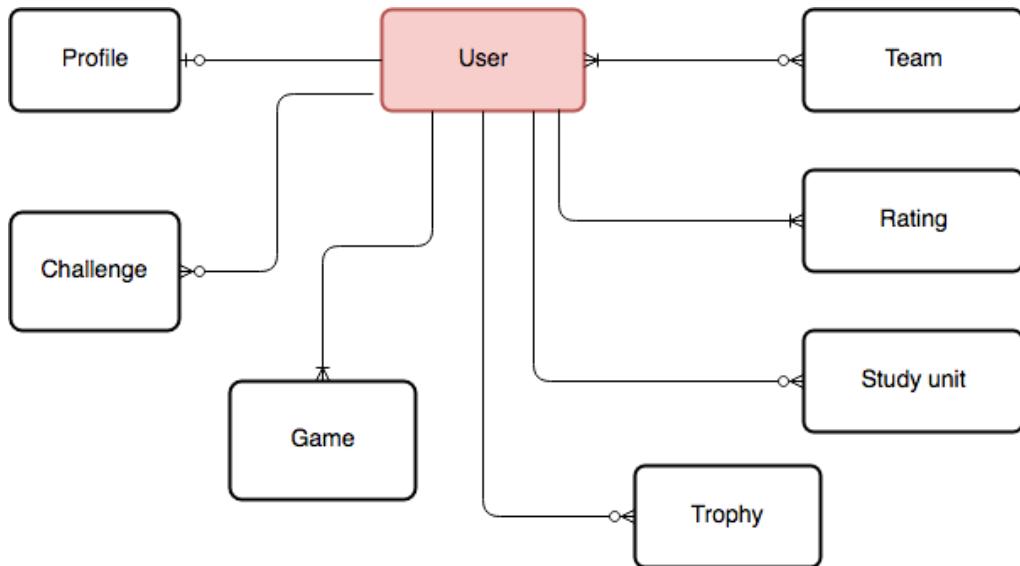
Figure 14.6: User data model

A user can have a profile (public information for others to see), can be challenged by multiple other users, took part in games, may have trophies, may have open study sessions, does have rankings for all the game

modes and may be part of multiple teams. We try to depict the common situation of an active user that has played before and teams which are not empty.

A major data-intensive feature of lichess is its shared analysis capability (called Study). A user can be part of multiple collaborative studies, which are collections of annotated (partial) games, used to analyze and learn specific games, openings and moves. These studies are saved in lichess database and they can be private (only visible to collaborators) or public.
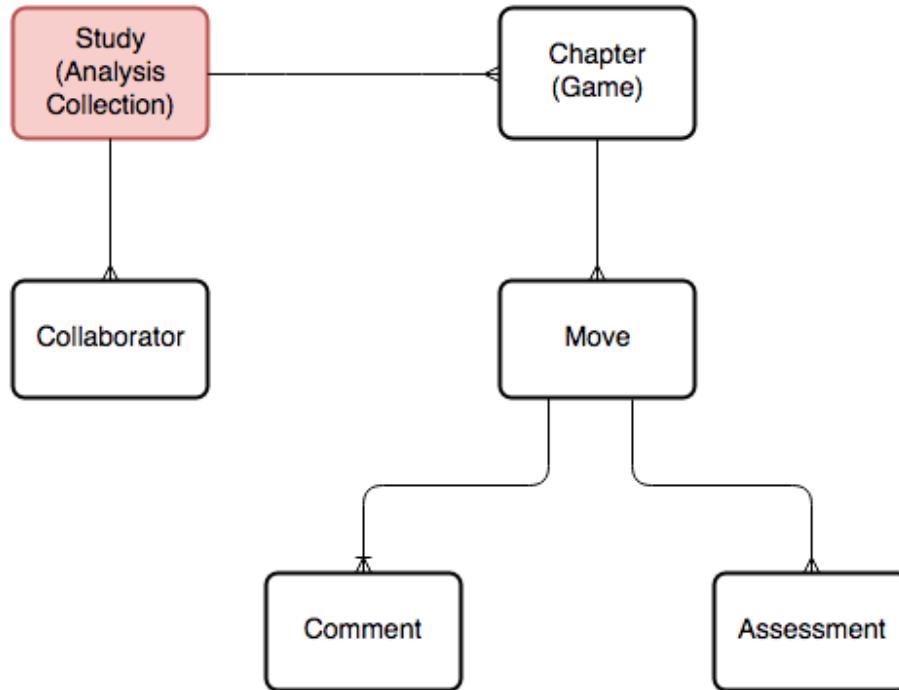
Figure 14.7: Study data model

The data structure for a single game is quite big with many members. All core chess related data is defined in the Game class inside the scalachess library. The Game class in the game module wraps this core data with administrative information such as the player IDs and a game ID, as well as time related metadata. A very compact data model omitting all smaller details can be seen in the following.

Notably the moves are stored in the popular PGN (portable game notation) format, which is used by many chess analytics programs.

14.5.4   Information Flow

The general information flow in lila is dictated by its nature as a web application. Information is generated, processed and persisted as a result of requests by the client (user).

Lila is organized into modules, seperated by responsibilties in terms of logic and data. The diagram shows the usual architecture for the non-utility modules (e.g game, tournament). A singleton object exposing
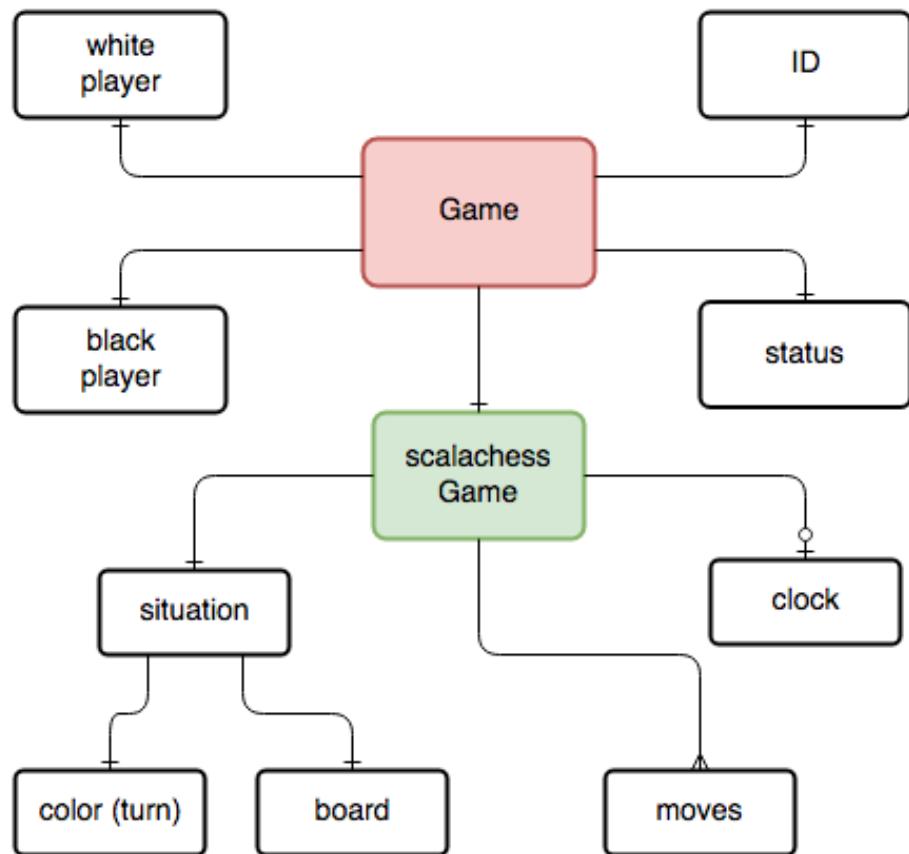
Figure 14.8: Game data model
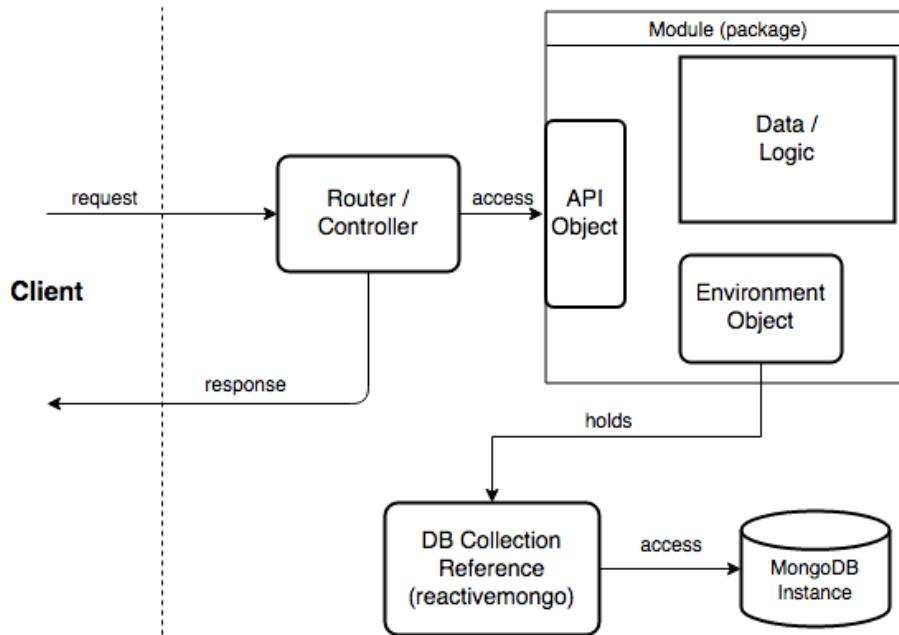
Figure 14.9: Information flow

an API defines operations that are performed by the respective controllers. General configuration and specifically references to the collections of data required within the module are collected in a singleton environment object. The datastructures corresponding to these collections are defined including conversion logic to BSON (the format used by MongoDb). Database access is only performed through the collection references and their API, which are defined by the reactivemongo library.

14.5.5   Ownership, Lifecycle and Privacy

Data ownership concerns the conceptual and physical distribution and authority on information. In the case of lila the authoritative data source is the central database which resides on servers operated by the project maintainer. Backups are made monthly and partly openly published.

Information lifecycle concerns the period of time data is stored until it is deleted (if not stored permanently). In the case of lila data storage is pretty much permanent, accounts can not be deleted, but only closed. This is due to the fact that accounts are associated with the games that the user played. These games are also permanently stored, so it does not make sense to allow account deletion if the information is to be kept in a consistent state [13].

As mentioned above all games played on lichess are published to the public. The games are published as monthly collections in the PGN (portable game notation) format, which is a widely used format for chess games.

---

[13]"Lichess privacy policy." [Online]. Available: https://lichess.org/privacy. [Accessed: 11-Apr-2019].

14.6 Conclusion

Lila is an impressive project in its scope and structure given that it has only one (main) maintainer. The most important stakeholders involved are the users and developers who also fulfill most of the other typical roles identified by Rozanski et al. The development process is very informal without extensive contribution guidelines. This is both a benefit and disadvantage, the project could attract more contributors with a clearer contribution guide, but with the current situation contributions are still made quite easily given the ususual fast and direct responses by the maintainer.

The project is structured very methodically according to features, but makes very little use of unit testing and other static analysis tools. This may speed up development at the cost of quality assurance. Technical debt is accumulating within the project, mainly in the form of code duplication or overly complex methods. At the moment no large impact on functionality and stability can be detected.

Lila is a data-intensive project processing massive amounts of chess games, tournaments and study sessions by more than a million users. It seems that currently there are no problems with the data persistence approach using a central document-store database. However given the growth of the website, there may be scalability issues in the future.

14.7 Appendix

List of merged pull requests for stakeholder analysis:

1. https://github.com/ornicar/lila/pull/1206
2. https://github.com/ornicar/lila/pull/892
3. https://github.com/ornicar/lila/pull/1900
4. https://github.com/ornicar/lila/pull/3981
5. https://github.com/ornicar/lila/pull/2701
6. https://github.com/ornicar/lila/pull/2229
7. https://github.com/ornicar/lila/pull/2169
8. https://github.com/ornicar/lila/pull/3046
9. https://github.com/ornicar/lila/pull/2742
10. https://github.com/ornicar/lila/pull/909

List of rejected pull requests:

1. https://github.com/ornicar/lila/pull/4591
2. https://github.com/ornicar/lila/pull/2115
3. https://github.com/ornicar/lila/pull/788
4. https://github.rozanski/ornicar/lila/pull/1510
5. https://github.com/ornicar/lila/pull/1511
6. https://github.com/ornicar/lila/pull/1338
7. https://github.com/ornicar/lila/pull/879
8. https://github.com/ornicar/lila/pull/1472
9. https://github.com/ornicar/lila/pull/2338
10. https://github.com/ornicar/lila/pull/3268

14.8   References / Footnotes

Chapter 15

MAPS.ME

By Henk Grent, Mark Haakman, Frenk van Mil, Casper Schröder.

15.1 Table of Contents

15.2   Introduction

MAPS.ME is an offline map navigation app developed for Android and iOS devices, built on top of crowd-sourced OpenStreetMap data. In November 2014 the app was acquired by the Mail.Ru group as part of the My.com brand. The application was originally closed source but opened to the public as of September 2015.

The purpose of this chapter is to assess the quality of MAPS.ME as an open source project on the basis of a software architecture standpoint, by analyzing several aspects such as the integration process, context, structure, and technical debt.

This chapter provides an overview of the MAPS.ME project. First, the chapter identifies the stakeholders involved. Next, the chapter describes the pull request procedure of the project and the integrators involved in this process. Furthermore, a context view, development view, and information view describe the architecture and design. The last part contains an analysis of the technical debt identified.

15.3 Stakeholders

By identifying the stakeholders related to MAPS.ME it is easier to choose priorities and identify potential constraints. Functioning mainly as a mobile app, MAPS.ME relies strongly on the size of their user base and the companies this userbase attracts. Descriptions of stakeholders are inspired by Rozanski and Woods [25].

15.3.0.1 Users

MAPS.ME's main feature of interest for end-users is its offline functionality. MAPS.ME provides high-quality, crowd-sourced maps for free. According to the MAPS.ME partners page [17], their audience consists out of people between the ages of 18 - 35, who are recreational travelers or business travelers.

15.3.0.2 Support staff

Support to other stakeholders is offered by means of e-mail contact and an online FAQ and knowledge base [18]. The support staff is assumed to be employed by MAPS.ME.

15.3.0.3 Suppliers

Supplier stakeholders build and/or supply the hardware, software, or infrastructure on which the system runs. There are a large number of external software dependencies. Two major stakeholders are Qt and C++. These dependencies are further identified in the context view. The app is deployed on both iOS and Android. As a result, they are dependent on Android and iOS. The terms and conditions of these stores [7; 2] can strongly influence the design choices MAPS.ME can make. Although the server focusses on offline navigation, it still requires OpenStreetMap to update the maps every once in a while.

15.3.0.4 Developers

The original MAPS.ME design and implementation were done by four people [13]. None of these developers appear to be contributing much, if any, code at this point [6]. Currently, there is a core team working for My.com (the owner company), situated in their office in Moscow, Russia.

After 2015, people we identified as most important developers are bykoianko, maksimandrianov, mpimenov, and rokuz. These developers seem to have more power than most other developers and will be further elaborated upon in the Integrators section.

15.3.0.5 Testers

Timofey Danshin has been doing tests and automation for the Mail.Ru Group since Dec 2015. However, he does not appear to be focused on MAPS.ME. Earlier in the project, there was no dedicated tester. Every contributor/maintainer is expected to provide test code for added features, according to the documentation. However, this requirement is rarely carried out in practice.

15.3.0.6 Missing roles

The other roles described in Rozanski and Woods [25], namely system administrators, acquirers, assessors, communicators, maintainers, production engineers and could not be identified. It is likely that these roles are either filled by someone from the Mail.ru group or are ignored completely. Another possibility is that integrators fill these roles, however, this is not clear from their GitHub activity.

15.3.1 Other stakeholders

Other stakeholders include stakeholders that were not mentioned by Rozanski and Woods [25], but which are still important entities that can influence MAPS.ME's position.

15.3.1.1 Competitors

Competitors include software products produced by other companies or developers which offer similar functionality. Software products freely providing access to world maps are common. Notable are Google Maps and Apple Maps. Both of these competitors offer limited offline access to maps on mobile devices.

15.3.1.2 GitHub contributors

GitHub contributors add to the project by creating issues, pull request, and reviewing pull requests. These contributors can fulfill a similar role as the developers. However, they are less informed as they cannot easily integrate with the core development team. They are also less powerful, as they need explicit approval from developers in the MAPS.ME core team.

15.3.1.3 Funders

Funders include any party that provides conditional or unconditional revenue to one or more contributors of MAPS.ME. There are a number of sources of income. The mobile application displays Google Ads [8]. For some fee, end-users can pay to hide advertisements, smaller companies can promote their company, and bigger companies can engage in partnerships [17]. Booking.com, for example, is a major partner of MAPS.ME.

15.3.2 Power interest grid

The power and interest of the significant stakeholder within the MAPS.ME context is visualized in *Figure 1*. They have been mapped from the viewpoint of the stakeholders themselves.

Overall Google's position does not allow it to assume a position that can be strongly detrimental to MAPS.ME. Google Ads does have power by means of providing ad revenue, however MAPS.ME has other sources of income as well. The app stores have the power to lower the search ranking of MAPS.ME, however, they currently do not appear to lower MAPS.ME in the search ranking as it even considered to be
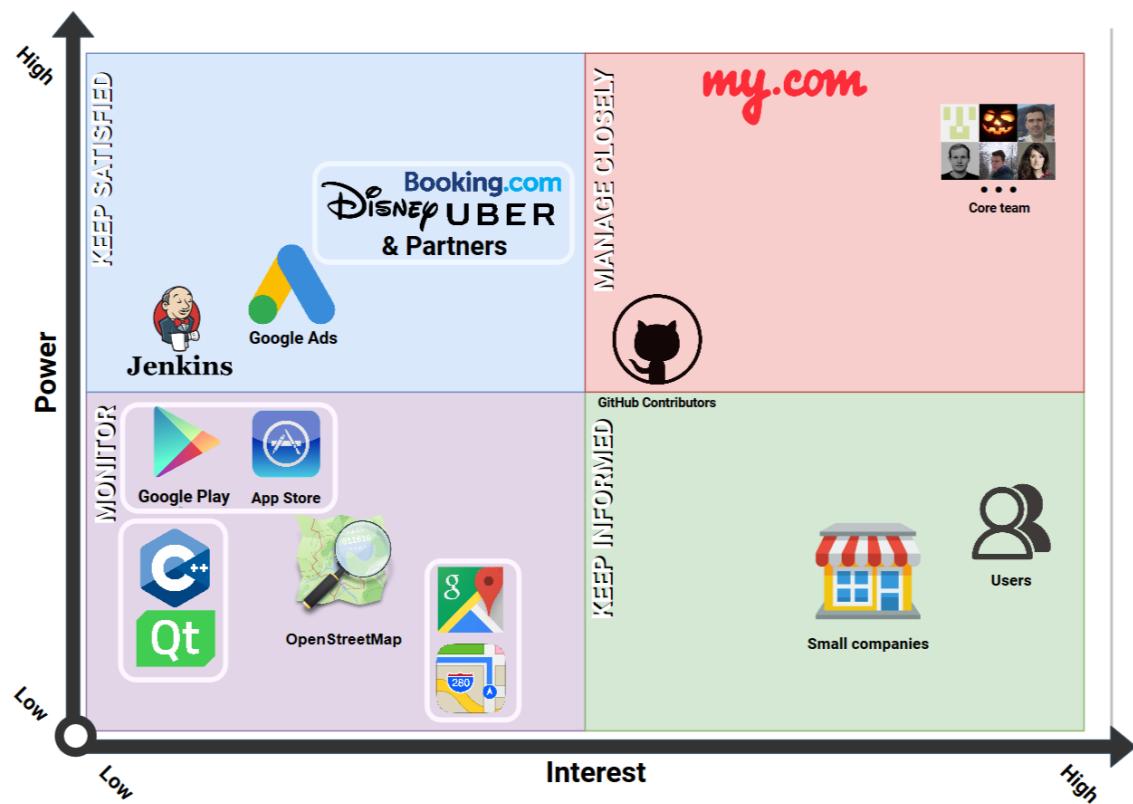
Figure 15.1: Figure 1: The power interest grids of significant stakeholders.

an 'Editor's Pick'. Finally, there is no connection between Google Maps and MAPS.ME, which results in it having low power.

All in all, MAPS.ME has established a relatively independent position in its market, in which their most significant concerns revolve around keeping their sources that generate revenue satisfied and maintaining their user base.

15.4   Pull Request Analysis

We analyzed 20 pull requests in order to gain insight into the decision-making process of the integrators involved in the project. We picked the 10 merged and rejected (closed) pull requests with the most comments, as these enabled us to give insight into the communication and workflow of the project. The entire analysis can be found in Appendix 1, but this section will provide the conclusions that were made after looking at these pull requests.

The 20 pull requests were all very similar in terms of communication. Lots of decisions are made in some other way than using GitHub communication. These pull requests are either merged or rejected without any clarification, or with a message, for example "discussed on Gitlab". Most of the discussion being done on the pull requests revolve around code quality on file level. It is rare to see anything else, and when it happens it is often optimization proposals.

The code quality is looked after very well, with one pull request having 800 comments discussing not much else. The proposed changes include fixing or improving code conventions, grammar, and redundancy in the code. Strangely, this does not include documentation of the code or corresponding comments. This gives the impression that the contributors care lots about code quality, but not documentation. Additionally, they rarely seem to question the choice of implementation. This might imply that decisions concerning implementation are made using other platforms of communication. On top of the disregard for documentation, there also seems to be a disregard for testing. Testing is often ignored in pull requests, or postponed initially and ignored afterward.

Bykoianko is the most active in pull requests, as such he appears to be the project leader. However, the Linkedin accounts of the MAPS.ME team reveal that Rokuz has the role of project leader and Bykoianko the role of senior developer.

A significant group of other contributors write their reviews in Russian. In contrast, all commit messages, comments, and language used in the code are English.

15.5   Integrators

Based on the analysis of pull requests, the main integrators of the project, and their decision-making strategies are described below. This analysis is key for understanding the interaction of people and developers within the project.

Roman Kuznetsov (Rokuz) Rokuz interacts in an almost product owner way. He reviews more than he contributes. When he opens a pull request, it is often without description and quickly merged after not more than one approval. If he approves a pull request, no other approvals are needed. His approvals are often given without any visible feedback on the contribution. Contact: N/A

Vladimir Byko-Ianko (Bykoianko) Bykoianko seems to be one of the most important people on the project. He either implements a feature or reviews it in depth. Important decisions on performance and how features get implemented are made by him. He is also one of the biggest contributors at the moment. Contact: N/A

Maxim Pimenov (Mpimenov) Mpimenov seems to be fulfilling the quality control role in the project. He comments on pull requests with mostly proposed style changes, or hard to miss bugs, which almost always get immediately fixed by the proposer of the change. Contact: N/A

Maksim Andrianov (Maksimandrianov) Maksimandrianov is a relatively new integrator of the project, however, is allowed to merge. He often partakes in the review process of features made by others. Contact: Maksimandrianov1@gmail.com

Tatiana Yan (Tatiana-yan), Vlad Mihaylenko (Vmihaylenko), and Gmoryes Tatiana-yan and Vmihaylenko, seem to fulfill the quality control role next to their developer role. Gmoryes is one of the new contributors to the project, having started on it half a year ago. The review process of his pull requests tend to be longer than others because more faults are present in his code. Contact: N/A, vxmihaylenko@gmail.com, and N/A.

15.6 Context View

The context view describes the relationships, dependencies, and interactions between the system and the environment [25]. The context view of MAPS.ME can be found in *Figure 2*. The elements in the context view are extracted based on stakeholders, build dependencies, development dependencies, and observing the project repository on GitHub.

MAPS.ME works differently than most popular navigation applications. Instead of using an internet connection to retrieve and show maps in real time, the application is fully offline. This appeals to a different user group. These users are often travelers or citizens that live in an area with a bad internet connection. This gives the application a valid competitive position compared to other map services, most importantly Google Maps and Apple Maps. This position is important because the app is used on Android and iOS, which are owned by Google and Apple respectively.

To properly build apps for these platforms, Java, Swift, and some third-party applications are used. HTML5 is used for the website and C++ for feature- and main development. The Android module is built with Gradle, the project uses GitHub for version control, and Jenkins as continuous integration service.

An important external entity is OpenStreetMap. OpenStreetMap is an external system to the project which provides the project with crucial map data. Map data of OpenStreetMap is crowdsourced from scratch and changes to maps can be made from within the MAPS.ME apps when an OpenStreetMap account is linked. Besides the points of interest provided by OpenStreetMap, Booking.com adds hotels to the maps, with functionality to directly book a room through the MAPS.ME app.
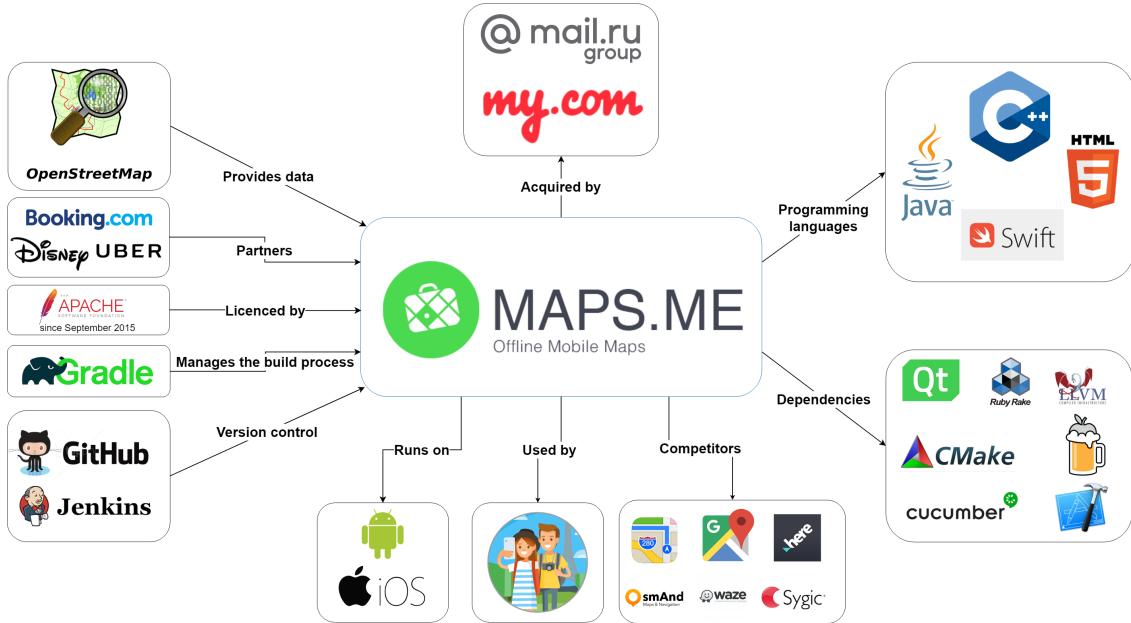
Figure 15.2: Figure 2: The context view of MAPS.ME

MAPS.ME is owned by My.com, which is part of the Mail.ru group. Maps.me used to be closed source and a paid service until it got bought by Mail.ru, after which it went open source. It now uses the Apache software license for open source projects.

15.6.1 Scope and Responsibilities

MAPS.ME is an offline navigation app. Its main responsibilities are showing map data properly without an internet connection, providing routing functionality for its users, and showing information on local shops, traffic, transport, and tourist information. Users can also use their GPS to see their location on the maps whilst being offline. The app covers the entire world, as it uses OpenStreetMap data. To prevent the app from taking an immense amount of storage on the device by storing the map files for the world, the separate map files for areas should be handpicked and downloaded by the user beforehand.

The app has some integration with parties such as Booking.com, to add functionality for users and create a revenue stream for my.com. The advertising focuses on travelers mostly and the app lets businesses advertise themselves inside it. The app is free to use, but if the users want an ad-free experience they can pay a subscription fee. It is only available for smartphones, so other more "classic" navigation devices and computers/laptops are outside of the scope.

15.7 Development view

The purpose of the development view is to provide insight into the code structure, dependencies and code standards [25]. This helps software engineers work with the system efficiently and in a way that does not violate technical integrity constraints. As of 2nd of March 2018, the repository contains about 46000 files and 5000 directories, totaling 7 GB. This is large enough to make orientation non-trivial.

15.7.1 Module structure

The 5000 directories of MAPS.ME can be structured in modules. The function of these modules and important dependencies between them are shown in the *Figure 3*. The module structure reveals how important components of MAPS.ME's architecture work together. No official module structure has been documented in the MAPS.ME repository.

The code is separated in different frames based on their functionality. The Maps frame includes the functionality related to creating and loading the map files (.mwm files) used by MAPS.ME. The Dependencies frame contains a broad generalization, as the so-called "3party" module contains 30 different third-party dependencies. Qt is shown separately, as it has its own folder in the main folder. All the dependencies and map files are used by code in the "Core" frame.

The End-user functionality provides functions like routing, searching for streets and businesses, and live traffic information. Business logic contains (among others) ads, code to process business partners, and a benchmark tool. The GUI contains everything needed to render and display the graphics of the app.

The core is compiled using Cmake. This creates resources from the separate functionalities in the core frame, which are used by the Android and iPhone projects. These projects create the final product in the form of apps. Some improvements can be made to this structure and data flow, which are discussed in Appendix 2.

15.7.2 Common design

Ideally, anyone who contributes to a software system contributes in a way that is consistent with the existing code base. This includes aspects such as a common coding style, standardizing test requirements, and the use of design patterns to improve cohesion. Unfortunately, the project's idea of a common design appears to be limited.

15.7.2.1 Common processing

The generator module is a map building tool used from a lot of other modules. According to the README file [14] "For development, use MAPS.ME Designer app along with its generator tool: these allow for quick rebuilding of style and symbols (. . .)". Furthermore, no explicit design patterns or contracts that should be adhered to are mentioned in the repository. Most of the functionality that is processed in a uniform manner appears to be incidental or as a result of the system's implicit constraints.
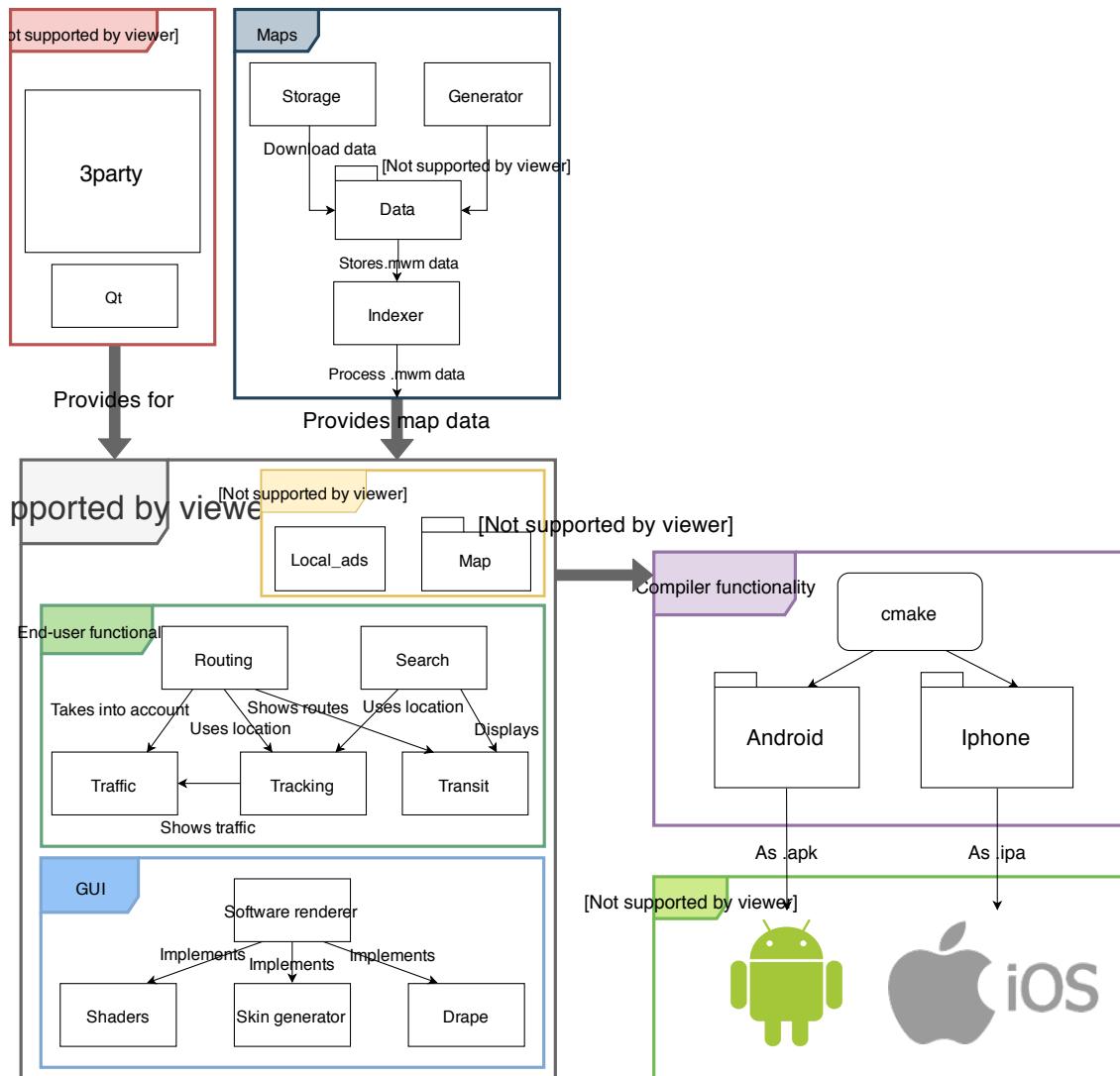
Figure 15.3: Figure 3: Module structure of MAPS.ME

15.7.2.2   Common language

Formally the MAPS.ME repository uses English for its documentation and code, whereas discussions on GitHub are often held in Russian [19]. The latter makes it hard for non-Russian speakers to participate.

15.7.2.3   Coding style

The coding style [15] has been extensively documented in the MAPS.ME docs. For C++ Google's coding standard [9] is used, with a number of exceptions. For the coding style, ClangFormat [4] is used with a specified configure file. On top of this, a piece of example code has been added.

For map files, MAPS.ME uses an adapted version of MapCSS [22]. A description of this format is not specified. However, a MAPS.ME specific designer app has been built which allows anyone to quickly build maps in a standardized way.

15.7.2.4   Standardization of testing

Documentation on the standardization of testing is very limited. Within the coding style document, it is mentioned that code should be covered with unit tests. It is also mentioned that third-party libraries should be avoided if they are not fully tested on the MAPS.ME platform. In practice, the amount of testing done is limited and not standardized.

15.7.3   Codeline model

The codeline model describes the source code structure, release process, configuration management, and build and testing approaches [25].

15.7.3.1   Source Code Structure

There are 49 folders in the root directory of the MAPS.ME project [29]. An overview with descriptions of the source code structure of MAPS.me is given in *Figure 4*. The descriptions are extended descriptions from CONTRIBUTIONS.md. One contribution to MAPS.ME by us was the addition of missing folder descriptions. As this repository consists of hardly any documentation and comments, it is hard to find out the purpose of each directory in the root folder. Therefore, not every change was accepted in our pull request #10513. Each white description in the figure was already present in the MAPS.ME documentation. The other colored descriptions were (trying to be) added in the pull request. Descriptions indicated in red were not accepted by Mpimenov [21], an author of the project. An elaboration on what information was missing was not given. Descriptions indicated in yellow were partially wrong, green descriptions are accepted and can now be found in the MAPS.ME documentation. Note that the "Core" described in this figure is different from the "Core" described in the Module structure. The Core in this figure was taken from the MAPS.ME documentation while the "Core" in the Module Structure was specified by us as it contains the core functionality.

pported by viewer]	api	\(undocumented and incomplete) external API of the application.\\
\<div style=	base	n: left">\some common coding functionality, like macros, logging, caches etc.\\ \</div>
	coding	\ I/O classes and data processing.\\
	drape	[Not supported by viewer]
\<div st	drape_frontend	ft">\scene and resource manager for the Drape library.\\ \</div>
	generator	[Not supported by viewer]
	geocoder	[Not supported by viewer]
	geometry	[Not supported by viewer]
	indexer	[Not supported by viewer]
	map	[Not supported by viewer]
\<div style=	platform	n: left">\platform abstraction classes: file paths, http requests, and location services.\\ \</div>
	routing	[Not supported by viewer]
routing_common		[Not supported by viewer]
	search	[Not supported by viewer]
\<div style=	std	n: left">\standard headers wrappers, for Boost, STL and C-rt.\\ \</div>
	storage	[Not supported by viewer]
	tracking	[Not supported by viewer]
	traffic	\real-time traffic information.\\
	transit	\ routing for public transport.\\
	ugc	\user generated content, such as reviews.\\

pported by viewer]	3party	\folder with 29 external libraries, some of which modified.\\
\<div style=	android	n: left">\Android UI. Views of the app and ads within the apps. Java code.\\ \</div>
	cmake	\CMake helper files.\\
\<div style=	data	n: left">\data files for the application: translations, maps, styles and country borders.\\ \</div>
	debian	[Not supported by viewer]
descriptions		\descriptions of features in different languages.\\
	editor	[Not supported by viewer]
tyle="text-align: l	feature_list	No documentation. This package does not seem to be consistent and no usage of the file could be found. It might ha
	installer	[Not supported by viewer]
	iphone	lot supported by viewer]
	kml	\manipulation of Google Earth KML files.\\
\<div style	local_ads	n: left">\ featured places on the map, for example shops.\\ \</div>
	mapshot	\generate screenshots of maps, specified by coordinates and zoom level.\\
	metrics	\ measuring user metrics.\\
\<div style	openlr	n: left">\ dynamic location referencing (cross referencing maps).*\\ \</div>
partners_api		\API for Booking.com, Facebook, and other partners.\\
	pyhelpers	[Not supported by viewer]
	qt	[Not supported by viewer]
	qt_tstfrm	[Not supported by viewer]
	shaders	[Not supported by viewer]
\<div st	skin_generator	ft">\a console app for building skin files with icons and symbols.\\ \</div>
software_renderer		[Not supported by viewer]
	stats	[Not supported by viewer]
	testing	[Not supported by viewer]
	tizen	[Not supported by viewer]
\<div style=	tools	n: left">\tools for various purposes e.g. building packages, testing, etc.\\ \</div>
\<div	track_analyzing	ft">\ analyzer for the generated tracks with the track generator.\\ \</div>
\<spa	track_generator	ian>Generate smooth tracks based on waypoints from KML.\\\ \
	xcode	[Not supported by viewer]

\<div style="text-align: left">\
\</div>

Figure 15.4: Figure 4: Source code structure of MAPS.ME

15.7.3.2 Build Approach

MAPS.ME has various build procedures. CMake is used as C++ compiler tool to create releases for Android and iOS. The build automation system Gradle is used for the Android version and the iOS version can be built within XCode. The project is developed on MacOS, so building on Linux based systems like Ubuntu will also easily work. However, building on Windows is not supported by the project.

15.7.3.3 Release Process

The source code is released on GitHub by merging open pull requests directly to the master branch. Since the real release of builds happens in the Google Play Store and Apple's App Store, failing builds on the master branch are not as critical as is the case for some other projects. MAPS.ME only had thirteen releases in 2018 [1]. This is a slow deployment cycle compared to other actively developed apps. This could be justified by the fact that resources, such as maps and interesting points on the maps are updated independently of the app. Map updates, with new routes and locations, can be downloaded from within the app. A timeline of the app releases in 2018 can be found in *Figure 5*. The year 2018 is chosen to give an overview of an entire year of releases. It is noticeable on the timeline that some releases are followed quickly by another release. That could suggest that they did ship some bugged releases, which needed to be patched quickly. The update notes on the Google Play Store do not show that these releases contain bug fixes.

Figure 15.5: Figure 5: Timeline of MAPS.ME releases in 2018 on the Google Play Store

15.7.3.4 Code management

MAPS.ME uses a Pull-based Software Development Model [10]. Contributors other than the authors have to fork the repository and create pull requests to merge their contributions into the project. Before someone can contribute to the project, a Contributor License Agreement has to be signed [16]. The authors themselves create pull requests but do not provide a description most of the time. If a description is provided, it is often in Russian. Jira is used as issue and bug tracking system but this tool is not available to the public. On some pull requests, the Continuous Integration system Jenkins is used. Jenkins has to be started manually by commenting on a pull request with "JTALL" (Jenkins Test All). In that case, Jenkins tries to build the project for Mac, Android, Linux, and iOS. The details of Jenkins build info cannot be viewed, as these are stored on a private website.

15.8   Information View

In this view, we will consider the information- storage, structure and quality. One of MAPS.ME's main functions is to make map data insightful for end-users. Therefore, we consider the maps and map enrichment data. The map data is used to generate the polygons and so-called 'features' which are present on the maps. The map enrichment contains files like translations, style sheets and country metadata.

15.8.1   Information storage

For maps, data is retrieved from OpenStreetMap database [23] using some download utility files in the storage module - as depicted in the module structure. This downloaded data is processed into map files ('.mwm') via the 'generate_mwm.sh' script, which are stored as flat files on the MAPS.ME server.

The map enrichment files are all included on GitHub and shipped with the app. They are stored in the corresponding formats. For key-values stores '.txt' and '.json' files are used.

15.8.2   Information structure

The information structure we will elaborate on revolves around map data, which uses 'map enrichment' files. The statistics are not related to the map and enrichment data.
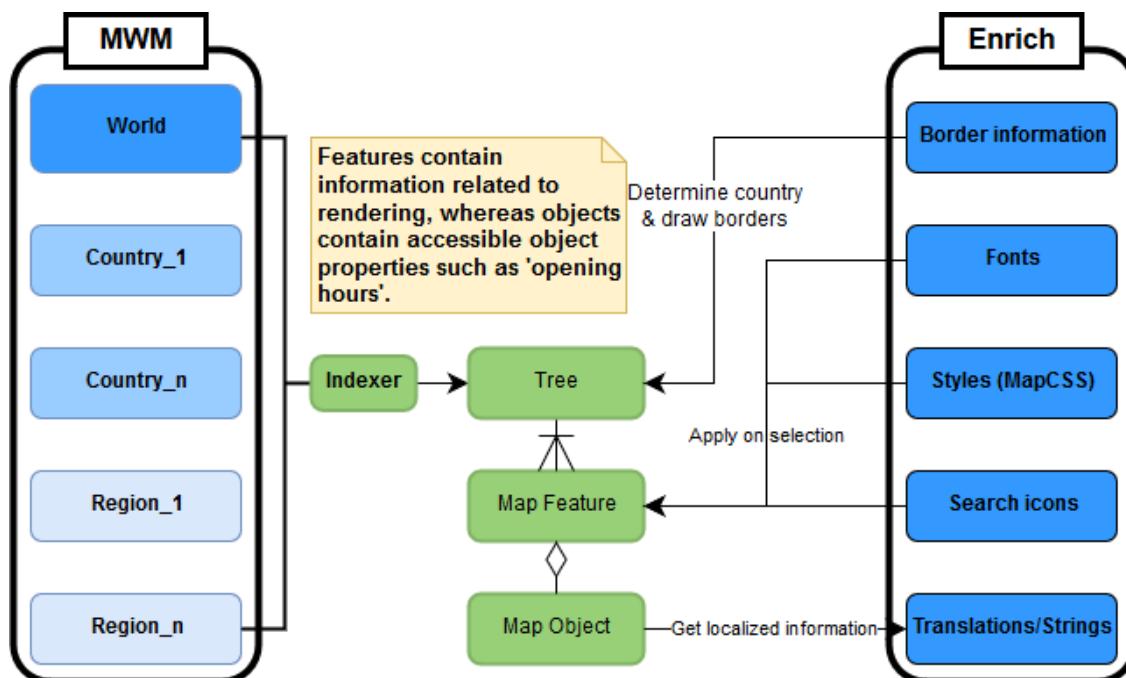
Figure 15.6: Figure 6: Information structure related to map files as static files and at runtime.

Figure 6 shows how static map- and enrichment files are used in creating the tree of map features that are used by MAPS.ME's core code. These trees are generated from .mwm files whose granularity depends on the location and level of zoom. If someone zooms in on a specific country, the .mwm file related to that country will be used. Only the '.mwm' file for the world view is included on GitHub and the mobile application by default. Specific map files can then be downloaded from within the app. This can be done by country or specific regions within this country.

The tree contains objects called map features. Each map feature consists of a specific type. There are numerous types, such as schools, industrial land, and pubs. These types contain geometry information, such as polygons, which is used to render them on the map. Some of these types may include a map object, which (generally) function as objects the user can interact with by means of the end-user functionality, as depicted in the module structure.

The features and objects in the tree use the enrichment data in different ways. The border information is used in many ways, most notably to determine the country features belong to and to draw borders. The fonts, style sheets, and search icons are applied to map features on a selection basis, similar to how CSS works. Finally, map objects use translation dictionaries to make strings location-specific.

15.8.3 Information quality

Information quality is warranted by using OpenStreetMap and having the project be accessible for contributions on GitHub. The map data used from OpenStreetMap is crowdsourced and open to adjustments. The data can always be updated to correspond with the real world, as OpenStreetMap updates its files every minute. This prevents users from getting annoyed by outdated data. This undermines the quality of the information slightly. However, if need be, this can be mitigated by releasing more often.

15.9 Technical Debt

Technical debt is the term introduced to indicate to nontechnical stakeholders the need for 'refactoring' [11]. From the original description, technical debt is described as "not quite right code which we postpone making it right"[5]. In this section, analysis tools are used to identify such technical debt. All analyses ran on the most recent version of the `master` branch at the time of writing this section (commit 1931f0f).

15.9.1 Maintainability

First, this section focusses on the maintainability of the project. Low maintainability could lead to surprises in cost and time when change is required [3]. One way to help identify maintainability of code is by the use of SonarQube. SonarQube is a static analysis tool capable of analyzing projects for bad practices, code smells, and more. Thanks to its extensiveness, it enables thorough analyses. As the MAPS.ME project is fairly big, the maintainability analysis will be narrowed down to the Android module.

15.9.1.1   SonarQube dashboard

From the SonarQube analysis dashboard 230 suspected bugs and 63 vulnerabilities can be found. Further-more, in *Figure 7* can be seen that the number of code smells is excessive (1.6k). Upon manual inspection, the bugs, vulnerabilities, and code smells are in many cases significant. Although there are indications that code smells do not always directly affect maintainability [26, 28], they are good indications of bad practices.

Figure 15.7: Figure 7: Dashboard overview of SonarQube of the Android app

Figure 8 shows the maintainability scores found by SonarQube. The majority of classes fall within 500 lines of code per class and a technical debt below 3.5 hours. The red cluster around seven hours of technical debt mostly contains Activity classes. Most red classes are caused by having far more parent classes than authorized (e.g. 12 parents where 5 is allowed).

The biggest outlier, with a technical debt of 1d 4h, is `MwmActivity.java`. By inspecting the class it becomes clear that this is the main activity of the application with many functionalities cramped into it and most of its code being abandoned code. This emphasizes the need for refactoring this file.

15.9.1.2   Bad practices

If we look at the complexity of methods in the whole project, some functions undoubtedly are refactoring candidates. For example, `normalize_unicode::NormalizeInPlace` has an NLOC (# lines of code) of 4571 and a Cyclomatic Complexity of 4234. This was only one of the most extreme cases, beyond doubt there are many more functions with a complexity even above a hundred. Most of these files are also identified (see *Figure 9*) by CodeScene, a static analysis tool capable of identifying refactoring candidates.

The technical debt on these files can become a concern if the code is not commented. If we look at the Android `/src` module, we see a comment/NLOC-ratio of 3.5% (measured with cloc). Compared to the average ratio found by Pascarella and Bacchelli[24], 40%, this is far lower than most prominent Java repositories. For C++ files, the comment ratio is 11.7%. Such low comment ratios make it harder to understand the code. Open source projects without adequate documentation are inherently at a disadvantage [12]. However, the core team does not seem to give a high priority to commenting or documentation. Contribution to the documentation was also partly rejected and the core team would rather postpone updating
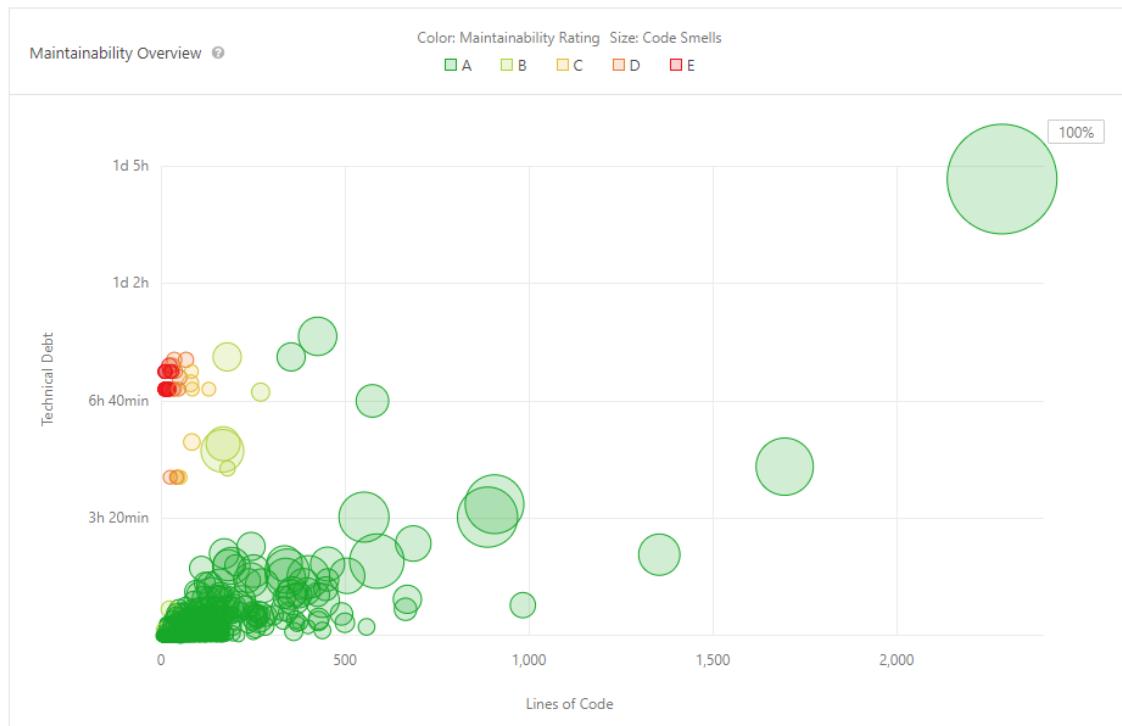
Figure 15.8: Figure 8: Maintainability graph of the Android app (from SonarQube)
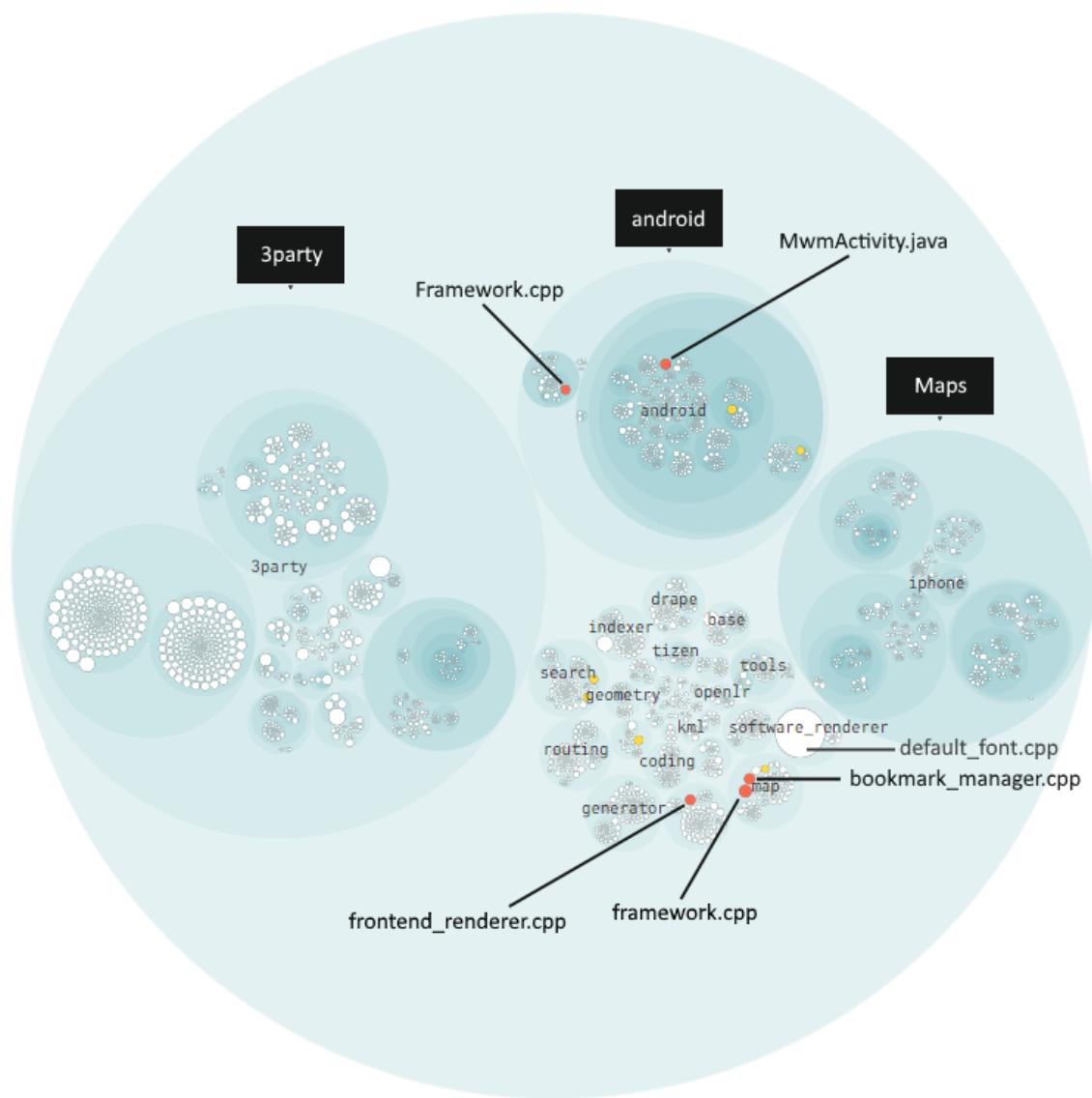
Figure 15.9: Figure 9: Refactoring candidates of MAPS.ME (from CodeScene)

the documentation (see #10513). The obvious solution to the technical debt in maintainability is to add documentation to the code and reduce the Cyclomatic Complexity in the identified refactoring candidates.

15.9.2 Testing debt

MAPS.ME does not use a standard testing framework. There are unit tests, but these can only be executed by their own test execution code [20]. This has some major disadvantages, which will be elaborated upon in this subsection.

Tests do not have IDE support, so executing tests can only be done from a terminal and results can only be viewed in a log file. This becomes a problem when a test fails, as it takes effort to find the cause. The script which executes tests, executes the test binaries separately, so there is no conclusive result of all tests. Failing tests are hidden between different modules in the generated log file.

The authors did not implement a way to measure code coverage into their testing module, so there is no easy way to tell which parts of the code are tested and which are not. This makes it very hard for us to discuss well- and less tested modules. When asking for more clarification on measuring code coverage in an issue #10555, we did not get any response as of writing this section.

To observe current testing behavior, we analyzed the last fifty pull requests to the project, which is a little more than a week of contributions. Of the fifty pull requests, only 12% changed test files. This indicates a significant testing debt.

Several actions should be taken to decrease further testing debt. As all main authors are developing in Xcode, the testing framework XCTest could be used to create and run unit tests, performance tests, and UI tests. This enables developers to debug tests, see code performance, measure code coverage, and automate testing.

15.9.3 Self-admitted technical debt

Technical debt is often introduced knowingly by developers, often indicated with "TODO"s, "FIXME"s or "BUG"s in the code. To find how many of these markers are present in the project, we used todo-show for Atom. This package allows to count and filter for the indications. The code contains a total of 2384 TODOs, 552 FIXMEs, and 19 BUGs.

A considerable part of the project is the code for the Android app. When building the Android app, the C++ compiler shows (600+) unhandled warnings. These warnings indicate potential issues. Either these warnings should be suppressed, or the code should be adjusted accordingly.

The total amount of markers in the code and the number of compile warnings give the impression that the code contains a hefty amount of self-admitted technical debt. As Storey et al. [27] have shown, developers believe that TODOs and other similar markers have a significant negative effect on the maintainability of the project. To avoid this technical debt, the developers should solve the warnings and identify the most important indicators.

15.9.4   Evolution of technical debt

To measure the evolution of technical debt, we analyzed the interactions of the integrators. The way they handle situations concerning technical debt can be an interesting benchmark. Often new features get merged without the mention of tests or even the added TODOs in the code. Occasionally, the integrators add tests for the most important functionality. In discussions around only a few pull requests, tests and comments are emphasized.

The last time a TODO was actually implemented was almost three years ago. The way of communication and implementation imply that the integrators ignore the technical debt of the project. They often add code containing technical debt and rarely discuss it. This causes the technical debt to keep growing and go unchecked.

15.9.5   Impact

With high Cyclomatic Complexity and the marginal documentation in the code, contributing to the project gets increasingly hard. The original author may be gone and, therefore, knowledge about functionality may be lost. Mistakes are more likely to be made, as confusion is dragged through the project. As the project has hardly any tests and the framework used for testing is incomplete, it is likely that many defects remain unnoticed. The defects and missing components that are noticed, e.g. with a TODO, are mostly ignored. After years of ignoring the indicators, the absence of their implementation can possibly cause extensibility problems or defects.

15.10   Conclusions

The conclusions reflect on the purpose of this chapter, which was to assess the quality of MAPS.ME with respect to the different software architecture viewpoints.

With respect to the context view and the stakeholders, MAPS.ME is a project which has a strong market position as an app. The emphasis on offline navigation allows it to separate itself from its competitors and its target audience is big enough to attract funders. Furthermore, MAPS.ME is not dependent on stakeholders which might have the intent to inhibit the project.

For the development view, MAPS.ME has a lot of room for improvement. The concerns of modules within the module structure are not always clearly separated. It is not easy to figure out the responsibility of modules, because the codeline structure is highly fragmented and there is little documentation. MAPS.ME could improve this aspect by including documentation about the data flow and the module structure. Also, better grouping the directories according to their functionality would improve MAPS.ME.

For the information view, MAPS.ME's dependence on OpenStreetMap allows it to keep its map data up-to-date. The information and related modules are well-structured. All in all, this makes MAPS.ME's approach for handling map data solid, yet relatively simple.

According to manual and automated analysis, we identified a significant technical debt. SonarQube revealed a high technical debt and low maintainability. A lot of this technical debt is self-admitted and has been shown to not reduce over time. The testing debt was not easy to measure. Having a uniform testing framework could help MAPS.ME control their tests better. In order to improve this aspect, the core team should become

aware of the consequences of high technical debt and how to deal with it. Static analysis tools could help with gaining such insight.

One particular theme among the different subsections of this chapter is the communication between the core team and the GitHub community. The project does not contain sufficient documentation to understand the code. The core team also does not respond hospitably when asked questions. Furthermore, the vast majority of the contributions are done by the core team without any reviews from external developers. As a result, it is hard to consider MAPS.ME a truly open source project.

15.11 References

[1] APKPure. 2019. MAPS.ME update version history for android - apk download. Retrieved March 22, 2019 from https://apkpure.com/maps-me-%E2%80%93-offline-map-and-travel-navigation/com.mapswithme.maps.pro/versions

[2] Apple. 2018. Apple media services terms and conditions. Retrieved March 22, 2019 from https://www.apple.com/legal/internet-services/itunes/us/terms.html

[3] T. Bakota, P. Hegedűs, G. Ladányi, P. Körtvélyesi, R. Ferenc, and T. Gyimóthy. 2012. A cost model based on software maintainability. In 2012 28th ieee international conference on software maintenance (icsm), 316–325. DOI:https://doi.org/10.1109/ICSM.2012.6405288

[4] Clang. Clang 9 documentation. Clang C Language Family Frontend for LLVM. Retrieved March 22, 2019 from https://clang.llvm.org/docs/ClangFormat.html

[5] Ward Cunningham. 1992. The wycash portfolio management system. SIGPLAN OOPS Mess. 4, 2 (December 1992), 29–30. DOI:https://doi.org/10.1145/157710.157715

[6] GitHub. Contributors to mapsme/omim. Retrieved March 22, 2019 from https://github.com/mapsme/omim/graphs/contributors

[7] Google. 2018. Google play terms of service. Retrieved March 22, 2019 from https://play.google.com/about/play-terms/index.html

[8] Google. MAPS.ME – offline map and travel navigation - apps on google play. Retrieved March 22, 2019 from https://play.google.com/store/apps/details?id=com.mapswithme.maps.pro

[9] Google. Google c style guide. Retrieved March 22, 2019 from https://google.github.io/styleguide/cppguide.html

[10] Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An exploratory study of the pull-based software development model. In Proceedings of the 36th international conference on software engineering, 345–355.

[11] P. Kruchten, R. L. Nord, and I. Ozkaya. 2012. Technical debt: From metaphor to theory and practice. IEEE Software 29, 6 (November 2012), 18–21. DOI:https://doi.org/10.1109/MS.2012.167

[12] Michelle Levesque. 2004. Fundamental issues with open source software development. First Monday 9, 4 (2004). DOI:https://doi.org/10.5210/fm.v9i4.1137

[13] MAPS.ME. 2015. Mapsme/omim contributors. Retrieved March 22, 2019 from https://github.com/mapsme/omim/blob/master/CONTRIBUTORS

[14] MAPS.ME. 2018. Mapsme/omim readme. GitHub mapsme/omim. Retrieved March 22, 2019 from https://github.com/mapsme/omim/blob/master/README.md

[15] MAPS.ME. 2018. Mapsme/omim coding style. Retrieved March 22, 2019 from https://github.com/mapsme/omim/blob/master/docs/CPP_STYLE.md

[16] MAPS.ME. 2019. Mapsme/omim contributing.md. Retrieved March 22, 2019 from https://github.com/mapsme/omim/blob/master/docs/CONTRIBUTING.md

[17] MAPS.ME. Partners. Partners. Retrieved March 22, 2019 from https://maps.me/partners/

[18] MAPS.ME. MAPS.ME – support. Retrieved March 22, 2019 from https://support.maps.me/hc/en-us

[19] MAPS.ME. Pull requests mapsme/omim. Retrieved March 22, 2019 from https://github.com/mapsme/omim/pulls

[20] MAPS.ME. Mapsme/omim testing tree github. Retrieved March 22, 2019 from https://github.com/mapsme/omim/tree/master/testing

[21] mpimenov. mapsme/omim Pull request #10513. 2019. Retrieved March 22, 2019 from https://github.com/mapsme/omim/pull/10513#pullrequestreview-213079484

[22] OpenStreetMap. MapCSS. About OpenStreetMap - OpenStreetMap Wiki. Retrieved March 22, 2019 from https://wiki.openstreetmap.org/wiki/MapCSS

[23] OpenStreetMaps. API. API - OpenStreetMap Wiki. Retrieved from https://wiki.openstreetmap.org/wiki/API

[24] L. Pascarella and A. Bacchelli. 2017. Classifying code comments in java open-source software systems. In 2017 ieee/acm 14th international conference on mining software repositories (msr), 227–237. DOI:https://doi.org/10.1109/MSR.2017.63

[25] Nick Rozanski and Eoin Woods. 2012. Software systems architecture: Working with stakeholders using viewpoints and perspectives. Addison-Wesley.

[26] D. I. K. Sjøberg, A. Yamashita, B. C. D. Anda, A. Mockus, and T. Dybå. 2013. Quantifying the effect of code smells on maintenance effort. IEEE Transactions on Software Engineering 39, 8 (August 2013), 1144–1156. DOI:https://doi.org/10.1109/TSE.2012.89

[27] Margaret-Anne Storey, Jody Ryall, R. Ian Bull, Del Myers, and Janice Singer. 2008. TODO or to bug: Exploring how task annotations play a role in the work practices of software developers. In Proceedings of the 30th international conference on software engineering (ICSE '08), 251–260. DOI:https://doi.org/10.1145/1368088.1368123

[28] Min Zhang, Tracy Hall, and Nathan Baddoo. 2011. Code bad smells: A review of current knowledge. Journal of Software Maintenance and Evolution: research and practice 23, 3 (2011), 179–202.

[29] MAPS.ME tree github. Retrieved March 22, 2019 from https://github.com/mapsme/omim/tree/master/

15.12 Appendices

15.12.1 Appendix 1: Full Analysis of Pull Requests

Pull request

Title + ID: [generator] Added wikipedia descriptions generation. #9811 Lifetime: Nov 9, 2018 - Dec 3, 2018 Components: generator/generator_tool, indexer Created by: Maksimandrianov Accepted/Rejected: Accepted Merged by: Mpimenov Addresses: Feature Issues: N/A Deprecates pull-request: N/A

This pull request adds a feature which allows the generation of descriptions for map elements on the basis of the Wikipedia API. Tests were included in this pull request. This is a feature which users of the system would want and the code was deemed to be of sufficient quality. A number of different people suggested minor changes to which Maksimandrianov responded with "++". There were also comments regarding more structural concerns, these were also responded to and addressed by Maksimandrianov.

Pull request Title + ID: [routing] Add speed cameras to routing + tests #9405 Lifetime: Sep 5, 2018 - Sep 17, 2018 Components: routing: speed_camera, map Created by: Gmoryes Accepted/Rejected: Accepted Merged by: Bykoianko Addresses: Feature Issues: N/A Deprecates pull-request: N/A

This feature adds a component displaying a UI icon and voice warning if a user approaches a speed camera. Tests were included in this pull request. This is a feature which users of the system would want and the code was deemed to be of sufficient quality. Bykoianko and Mpimenov gave the most feedback to which Gmoryes responded with 'done' or replied with a longer response which resulted in some discussion. The feedback consisted of a mix of structural feedback and some minor changes.

Pull request Title + ID: [routing] Fix crash because of transit id #10181 Lifetime: Jan 9, 2019 - Jan 10, 2019 Components: routing:corss_mwm_index_graph.hpp (Single file) Created by: Gmoryes Accepted/Rejected: Rejected Closed by: Gmoryes Addresses: Bugfix Issues: N/A Deprecates pull-request: N/A

This PR attempts to fix one equality assignment related to segment geometry. Closed because the integration tests fail and a fix was released in another version. Gmoryes and Bykoianko discussed the proposed fix in detail. In this discussion Bykoianko mentioned an alternative way to fix the issue, which involved using different methods touching different parts of the program. Bykoianko also explained why this might be beneficial.

Pull request Title + ID: Persian translation #9558 Lifetime: Sep 29, 2018 - Oct 8, 2018 Components: iphone/Maps/LocalizedStrings, data/strings Created by: Mnameghi Accepted/Rejected: Rejected Closed by: Mpimenov Addresses: Translation Issues: N/A Deprecated by pull-request: #9360

This PR adds Persian translations to MAPS.ME. Closed because there were some minor issues which were addressed in a different PR (#9630). This different PR, made by Tatiana-yan, was merged. A number of people said it looked good and thanked Mnameghi. Mpimenov gave a lot of feedback in terms of some characters or quotes which should be removed.

Pull request Title + ID: [generator] Region kv iso3166-1 #9349 Lifetime: Aug 23, 2018 - Sep 4, 2018 Components: generator: region files Created by: Maksimandrianov Accepted/Rejected: Closed Closed by: Maksimandrianov Addresses: Feature Issues: N/A Deprecated by pull-request: #9393

Adds regions to the generator. After not a lot of substantial suggestions, Tatiana-yan suggested a rebase after which Maksimandrianov closed it without further explanation. A pull request addressing the same issue was opened on 4 September 2018. This pull request was merged, however no mention to #9393 was ever given.

Pull request Title + ID: [generator] Region kv #9340 Lifetime: Aug 21, 2018 - Sep 3, 2018 Components: Generator/regions Created by: Maksimandrianov Accepted/Rejected: Accepted Merged by: Mpimenov Addresses: Feature Issues: N/A Deprecates pull-request: #9349

This pull request adds regions to the generator such as city, town, village, suburb, and neighborhood. The three reviewers of this pull request are Mpimenov, Tatiana-yan, and Syershov. There is a discussion going on if a certain third party application should be included in the codebase or not. There are also some grammatical and format errors addressed. Furthermore, code efficiency is improved after reviewing. Also, tests are added after the pressure of the reviewers. The pull request is accepted after a lot of conversation (377 comments in total), where code quality is vastly improved and bugs are solved together.

Pull request Title + ID: [routing] SpeedCameras Core and ios interface #9888 Lifetime: Nov 22, 2018 - Nov 30, 2018 Components: Implementation and interface of speed cameras on iOS Created by: Gmoryes Accepted/Rejected: Accepted Merged by: Bykoianko Addresses: Feature Issues: N/A Deprecates pull-request: N/A

This pull request has no description. It adds speed cameras to the interface of iOS and also adds the implementation of speed cameras. The reviewers Vmihaylenko and Bykoianko are addressing format and style improvements. It is finally approved after a conversation of 202 comments. Many inconsistencies, style issues, and small refactors are addressed.

Pull request Title + ID: [DNM][traffic] Switching off temp blocking traffic jam for Russia. #10060 Lifetime: Dec 13, 2018 - Dec 14, 2018 Components: traffic Created by: Bykoianko Accepted/Rejected: Rejected Rejected by: Bykoianko Addresses: Bugfix Issues: N/A Deprecates pull-request: N/A Deprecated by pull-request: #10078

This pull request is closed quickly because Burivuh wants that the code is added to master directly. The new pull request #10078 is discussed later on.

Pull request Title + ID: [DNM] http thread refactoring #10043 Lifetime: Dec 11, 2018 - Dec 12, 2018 Components: http_thread Created by: Beloal Accepted/Rejected: Rejected Closed by: Beloal Addresses: Refactoring Issues: N/A Deprecates pull-request: N/A

This pull request is a refactoring of HTTP thread classes. The refactoring is said to be bad by Maksimandrianov, Mpimenov, and Rokuz, as the refactoring introduces a lot of typecasting. The author of the pull request does not agree with this and closes the pull request as he thinks there is no further reason to discuss this matter. A similar solution intended by this refactor is implemented in pull request #10047 and self-merged by Beloal, this time with changes accepted by Rokuz and Vmihaylenko.

Pull request Title + ID: Remove duplicates from strings.xml #9789 Lifetime: Nov 4, 2018 - Nov 6, 2018 Components: strings Created by: Dimqua Accepted/Rejected: Rejected Rejected by: Rokuz Addresses: Translation Issues: N/A Deprecates pull-request: N/A Deprecated by pull-request: #9768

This is a small pull request which removes a duplicate string from the translations for the Android app. The pull request is closed by Rokuz because this particular string is already removed in an earlier pull request #9768.

Pull request Title + ID: Maxspeed section. #9697 Lifetime: Oct 18, 2018 - Nov 23, 2018 Components: generator maxspeed, routing Created by: Bykoianko Accepted/Rejected: Accepted Merged by: Mpimenov

Addresses: Feature Issues: #9697 Deprecates pull-request: N/A

The pull request adds a section to the application which holds information on speed limits of roads, saved with a "maxspeed" tag. The purpose seems to be the addition of a major feature. This pull request is the first pull request containing code related to this feature.

The process starts with the initiator (Bykoianko) asking for reviews of three other contributors: Vmihaylenko, Mpimenov, and Maksimandrianov. A different contributor joins in and discusses the implementation of a very specific method in the code, resulting in the initiator changing the code as proposed. All of the invited contributors propose several optimization changes, which are quickly accepted and implemented by the initiator. Mpimenov seems to adhere to a form of commenting which is as follows: s/x/y/, where the s/.../ implies x should be changed to y. Tens of bugs seem to be fixed by other contributors proposing changes. After many bugfixes, the reviewers approve the pull request one by one with no further clarification except for "LGTM" (looks good to me).

Noteworthy is that the initiator seems to use triggers for automated checks of the code. One of such triggers is "JTA", of which the purpose is unclear.

Pull request Title + ID: New IndexGraphStarterJoints, JointSegment, jumps between segments. #9857 Lifetime: Nov 16, 2018 - Jan 23, 2019 Components: routing Created by: Gmoryes Accepted/Rejected: Accepted Merged by: Bykoianko Addresses: Feature Issues: #9857 Deprecates pull-request: N/A

The pull request adds a part of a new feature, namely routing based on joints in an effort to improve routing.

The review process starts off with a "JTA" comment by Bykoianko, who joins the review process. He follows by pointing out several flaws in the implementation, and why it is not what he expected when proposing the feature. Some optimization changes are denied by Gmoryes, the initiator of the project because the time saved is not worth the amount of work. Both Bykoianko and Vmihaylenko post various strict comments on the language of the code comments, which need to be switched from Russian to English. After all these changes, and after running integration tests, Bykoianko decides it is good enough and merges the commit.

Pull request Title + ID: Added geo objects translator. #9592 Lifetime: on Oct 3, 2018 - Oct 5, 2018 Components: generator Created by: Maksimandrianov Accepted/Rejected: Accepted Merged by: Mpimenov Addresses: feature Issues: #9592 Deprecates pull-request: N/A

The pull request is part 1 of 2 and contains a translator that collects houses and points of interest (poi) and a collector that collects houses and poi without an address. This effort is made to increase the number of addresses and poi the app can handle, thus increasing its functionality.

Maksimandrianov starts off by requesting a review from Tatiana-yan, Mpimenov and Syershov. Tatiana-yan immediately asks for tests of the functionality, which seem to go ignored. Mpimenov follows up by suggesting many bugfixes and style corrections. The other contributors follow with similar suggestions. Tatiana-yan asks some more in-depth questions, which are either followed by a revision of the code, or further clarification. She seems to be the main reviewer for this pull request. Short description of interaction/discussion between users. Syershov seems to approve the pull request earlier than any other reviewers, with no apparent reason. After some small changes and a few bugfixes, the other reviewers also approve the pull request.

Pull request Title + ID: Add speed camera (ROUTING) #9331 Lifetime: Aug 17, 2018 - Sep 5, 2018 Components: routing Created by: Gmoryes Accepted/Rejected: Rejected Closed by: Gmoryes Addresses: Feature Issues: N/A Deprecated by pull-request: #9330

The pull request is made without any description, however, the title implies it adds a new feature, namely speed cameras. Bykoianko, Vmihaylenko, and Tatiana-yan propose several style changes and bugfixes. Bykoianko notes that there is something wrong with the way GitHub shows the changes. Some more discussion is done on aspects of the new feature, e.g. the notification to the user when a speed camera is close. The pull request is then closed, because as Gmoryes states: "GitHub does not correctly show the changes that already exist in master". It seems the code in this pull request is still accepted and merged in another pull request, namely #9330.

Pull request Title + ID: CityBoundariesChecker #9346 Lifetime: Aug 23, 2018 - Aug 27, 2018 Components: routing Created by: Bykoianko Accepted/Rejected: Rejected Closed by: Bykoianko Addresses: Feature Issues: N/A Deprecated by pull-request: #9361

Bykoianko opens the pull request with a new feature which checks the boundaries of cities. He asks Rokuz, Mpimenov, and Tatiana-yan to review it. Some discussion on the functionality is done, mostly by Bykoianko and Tatiana-yan. Mpimenov proposes several bugfixes and style changes. Rokuz approves the changes without any interaction. Eventually, Bykoianko decides the approach of the implementation is not satisfactory, due to performance reasons. The benchmark shows the load times on android increase by 13 seconds due to this feature. Bykoianko closes the pull requests and redirects the feedback to pull request # 9361, which implements a similar feature in a different way.

Pull request Title + ID: [routing] Add speed camera section (GENERATOR) #9330 Lifetime: Aug 17, 2018 - Sep 5, 2018 Components: routing, routing/speed_camera, generator, translator Created by: Gmoryes Accepted/Rejected: Accepted Merged by: Bykoianko Addresses: Feature Issues: N/A Deprecates pull-request: N/A

The pull requests add the option to have speed camera indications in the routing of the MAPS.ME client. The pull request is a big pull request as it affects 49 different files with around 2000 touched lines. The majority of these affected files were either in the routing package or in the generator package.

Interestingly, the author did not feel the need to explain this feature in his pull request. Moreover, his single commit affecting around 2000 lines also did not have any explanations for his changes. Of all of his changes, a vast majority was added without any comments. Still, the new feature was merged to the master without any complaints about the commenting or committing behavior. Therefore, it seems with this pull requests that none of the reviewers feel the need for any comments for future developers.

Almost all the changes that were requested were mostly concerning the coding style. For example, the order of imports. The type of remarks was similar for all three reviewers (i.e. `Tatiana-yan`, `Bykoianko`, and `Mpimenov`). None of the reviewers mentioned the size of the pull request or the number of files and components that were touched by it.

The need for the pull request becomes clear when one analyzes some of the competitors (like Google Maps and TomTom), who provide a similar feature for speed cameras. Therefore, the feature becomes clear in the context of a navigation system, where a driver would like to know where speed cameras are. However, the acceptance of the pull request leaves some doubt to the overall code quality of the project, as a clear lack of documentation leaves doubt to the design decisions made.

Pull request Title + ID: Indexed AStar routing #4672 Lifetime: Nov 7, 2016 - Nov 26, 2016 Components: routing, routing/geometry Created by: Dobriy-eeh Accepted/Rejected: Accepted Merged by: Bykoianko Addresses: Feature Issues: N/A Deprecates pull-request: N/A

In this pull request, the famous A* algorithm for routing is implemented with the support of indexing to improve the performance of the algorithm. To do so, most of the routing files were affected. In the pull

request around 2000 lines were changed of which a significant portion for the addition of new tests.

From all of the 729 comments in the full discussion of the pull request, only three discussions (with a total of 13 comments) were questions (and answers) covering the actual working of the implementation of the algorithm itself. All other comments concerned the code quality, code conventions, missing comments, or missing tests.

Pull request Title + ID: [routing] Applying restrictions on index graph #4817 Lifetime: Nov 26, 2016 - Mar 6, 2017 Components: routing Created by: Bykoianko Accepted/Rejected: Rejected Rejected by: Bykoianko Addresses: Feature Issues: N/A Deprecates pull-request: #4998

In this pull request, Bykoianko implements the imposition of restriction on the index graph. This is used to improve the algorithm for route generation. In the pull request, Bykoianko ensures that the reviewers understand his changes by giving examples.

Interestingly, none of the reviewers question the choices in the implementations or the working of the algorithm. Although the implementation is an advanced implementation, the only remarks that are made have to do with the code quality. This raises the question if the implementation is discussed outside the pull request or that no reviewer takes the time to question the algorithm.

In the end, the pull request is suddenly closed with a message that a dynamic restriction was implemented instead. This suggests that there was a discussion going on about the best algorithm to implement but this discussion is nowhere to be found. The alternative pull request (i.e. #4998), that was accepted, also does not contain a discussion nor does it contain a single review. Therefore, the review process is must have been done somewhere else. This could be an external application or in person in Moscow, where the MAPS.ME office is located.

Pull request Title + ID: [descriptions] Description section serializer and deserializer added. #9783 Lifetime: Nov 2, 2018 - Nov 23, 2018 Components: descriptions Created by: Darina Accepted/Rejected: Accepted Merged by: Mpimenov Addresses: Feature Issues: N/A Deprecates pull-request: N/A

The descriptions that can be added to elements in the MAPS.ME app can, with this new feature, be serialized and deserialized. This feature is used for the language system that is built into the application to support multiple languages.

As most pull requests, the vast majority of the comments are about the code quality. Interestingly, Darina does not accept all comments right-away. The pull requests contain discussions where Darina does not accept the review that was given and requires the reviewer to further explain his or her requests.

Also, this pull requests shows some evidence of discussions outside GitHub. Darina mentions that she discussed some of the implementation choices with Mpimenov, while there are no indications of discussion on GitHub of the subject.

Bykoianko discovered that the implementation lacked tests, so he asked Darina to add some. She, however, asked to postpone it for a new pull request which was accepted. Multiple reviewers agreed with this decision, which is arguably a bad decision. The chance that the tests will never be added is increased with the decision. Eventually, the pull request is merged with a request for a few changes in a future, undecided, pull request.

Pull request Title + ID: [DNM][traffic] Switching off temp blocking traffic jam for Russia. #10078 Lifetime: Dec 14, 2018 - Jan 18, 2018 Components: traffic Created by: Bykoianko Accepted/Rejected: Rejected Rejected by: Bykoianko Addresses: Feature Issues: N/A Deprecates pull-request: #10060 Deprecated by pull-request: #10210

According to Bykoianko, a temporary switch-off is required to not show temporary traffic blocking due to traffic jams. The feature only supports traffic jams in Russia.

The discussion for the pull request is short and abruptly stopped with the announcement that the pull request will be closed for an alternative. The decision was made to shift the implementation, which was client-side, to a server-side implementation. Three weeks later the alternative was merged. Again, evidence was shown that the developers discuss their concerns outside the pull request on GitHub. Bykoianko posted an URL to a Gitlab, which we could not access. The Gitlab is hosted by a Mail.ru server, which is the owning company of the project. It could, therefore, mean that the Gitlab is a private platform for discussions within the company.

15.12.2   Appendix 2: Possible architecture improvements

When looking at the top level (highly abstract) architecture, some aspects can be improved. For example, the module "3party" contains lots of third party dependencies. These could be implemented in a more structured way by creating third party handler classes (or interfaces) for each type of third party dependency. There could be one for GUI dependencies, one for map data dependencies, etc.

The other parts that are open for improvement are the part we call "Business logic" and the coding module. The "Business logic" part contains a module called "map". This "map" contains code for all sorts of functionalities, like booking information, gps tracking , cloud storage, user markers, etc. All this code seems to be dumped in this folder with no coherent structure whatsoever. This module could be split up for the different functionalities, and a neat tree like structure could be made for them.

The overall data flow of the project is unclear, as all resources are compiled by cmake, after which they are used by the subprojects for differing platforms. This could be changed, or documentation could be improved to make this clear for new developers.

Chapter 16

pandas | Python Data Analysis Library

Shaad Alaka   Max Lopes Cunha   Max van Deursen   Jop Vermeer

16.1 Table of Contents

16.2   Introduction

Pandas is an open source data analysis toolkit for Python which has enabled millions of users to perform complex data analysis in a relatively simple way. According to its creator, Wes McKinney, the package is meant to accelerate the data analysis workflow and reduce the need for people to deal with technical issues. It is currently used all over the world, with 5-10 million users worldwide. This chapter contains a collection of views and perspectives on pandas' architecture.

The development of pandas was started by Wes McKinney around 2008, who at the time was employed at AQR Capital Management. Wes sought to create a toolkit that could provide an intuitive and highly performant way to manipulate data in commonly used formats. Predating pandas, data analysis required custom libraries to merge and combine different data formats [1].

Filling this niche helped pandas jump-start the adoption of Python in the data science and data analysis communities. Since 2012, the popularity of Python and pandas has been increasing rapidly, in part thanks to the pandas project [2]. As seen in Figure 1, the amount of contributions spiked around this time as well, attracting a few loyal contributors over the years.
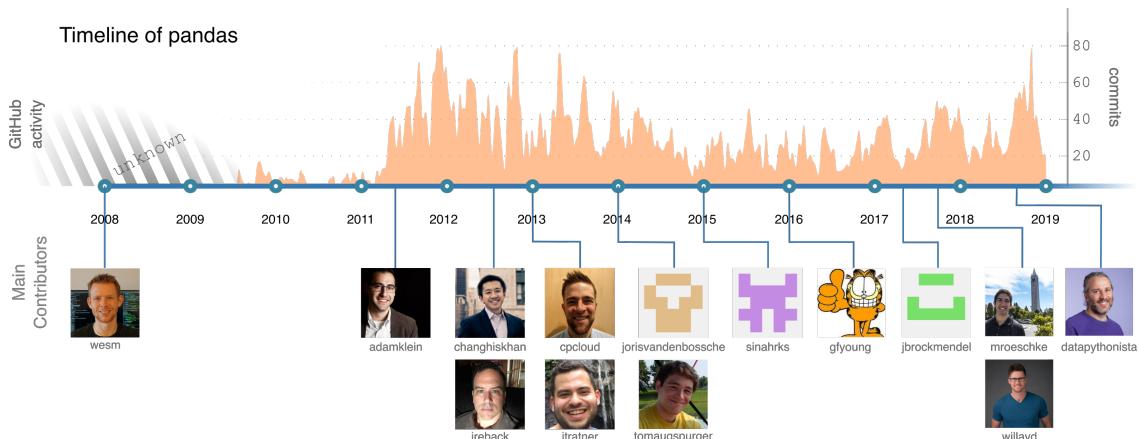
Figure 1: An overview of the history of pandas and its main contributors.

16.3   Stakeholders

To provide insight into the stakeholders of pandas, we adapt the categories defined by Rozanski and Woods [3] in a way we believe is suitable for pandas. Afterwards, we provide a power-interest grid to visualize the relative importance and interest of different stakeholders in the project.

---

[1] Tobias Macey (Host). Pandas with Jeff Reback - Episode 98 [Audio podcast]. https://www.pythonpodcast.com/episode-98-pandas-with-jeff-reback/. Accessed on 28 March 2019.

[2] Dan Kopf. Meet the man behind the most important tool in data science. https://qz.com/1126615/the-story-of-the-most-important-tool-in-data-science/. Accessed on 28 March 2019.

[3] Nick Rozanski and Eóin Woods. 2012. *Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives. Addison-Wesley Professional.*

16.3.1 Acquirers

Rozanski and Woods describe the acquirers of a product as stakeholders which oversee the procurement of the system [4]. Although pandas is now open source, it started as an in-house application at AQR Capital Management in 2008 [5]. Since 2015, NumFOCUS serves as the only fiscal sponsor of pandas [6].

16.3.2 Assessors

Assessors oversee conformance of the project to standards and legal regulations [7]. Pandas complies to the BSD license [8]. The legal conformance of the library itself is overseen by the pandas core development team.

16.3.3 Contributors

Since pandas is an open source software project, it is developed and maintained by contributors from all over the world. We observed two categories: community and enterprise contributors. Enterprise contributors distinguish themselves by explicit affiliation with a company, often tied with financial compensation.

Every contributor should comply to the contribution guide set by pandas [9], which mentions that contributors should provide tests and documentation when implementing new features. Contributors are free to work on fixing relevant issues or new features.

The pandas governance also defines core team members; a group of contributors who contributed considerably to pandas and together ensure the long-time well-being of the project [10]. Wes McKinney fulfills the role of chair and Benevolent Dictator for Life, giving him authority to make final decisions about pandas. The complete team is listed in Appendix A. From the core team, we identified people which are active on recent issues and are therefore suitable to contact: - Jeff Reback - Joris van den Bossche - Mark Garcia - Tom Augspurger - William Ayd

We contacted Joris van den Bossche about the pandas development team, leading to the following PR. Jeff Reback and Wes McKinney were already extensively interviewed [11] [12], answering all our questions, which made contacting them unnecessary.

---

[4]Nick Rozanski and Eóin Woods. 2012. Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives. *Addison-Wesley Professional.*

[5]pandas. History of Development. https://pandas.pydata.org/community.html. Accessed On: 25 February 2019.

[6]pandas. Main Governance Document. https://github.com/pandas-dev/pandas-governance/blob/master/governance.md. Accessed On: 25 February 2019.

[7]Nick Rozanski and Eóin Woods. 2012. Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives. *Addison-Wesley Professional.*

[8]pandas. Package overview. https://pandas.pydata.org/pandas-docs/stable/getting_started/overview.html. Accessed On: 28 February 2019.

[9]pandas. Contributing to pandas. https://pandas-docs.github.io/pandas-docs-travis/development/contributing.html. Accessed On: 25 February 2019.

[10]pandas. Main Governance Document. https://github.com/pandas-dev/pandas-governance/blob/master/governance.md. Accessed On: 25 February 2019.

[11]Tobias Macey (Host). Pandas with Jeff Reback - Episode 98 [Audio podcast]. https://www.pythonpodcast.com/episode-98-pandas-with-jeff-reback/. Accessed on 28 March 2019.

[12]Tobias Macey (Host). Wes McKinney's Career In Python For Data Analysis - Episode 203. https://www.pythonpodcast.com/wes-mckinney-python-for-data-analysis-episode-203/ Accessed on 4 April 2019.

Jeff Reback, Tom Augspurger and William Ayd can also be regarded as integrators as they are the ones labelling most issues and reviewing/merging PRs.

Contributors can be categorized as either developer or maintainer, depending on whether they worked on features or bug-fixes [13]. Moreover, since contributions to pandas include documentation and tests as well, each contributor is a communicator and tester as well.

16.3.4   Suppliers

Rozanski and Woods state that suppliers provide the software on which the product will run, or provide specialized staff for development or operation [14]. We distinguish between Software and Development suppliers:

- *Software Suppliers*: These suppliers provide software which is used when running pandas.
- *Development Suppliers*: These suppliers provide a service which is used in the development of pandas.

A full list of suppliers can be found in Appendix B.

16.3.5   Support Staff

Pandas does not have staff dedicated to support users with issues. However, they can receive help from the community in the following ways: - Post a question on StackOverflow, where other users can answer the question. - Consult the pandas documentation. Pandas provides users with an extensive documentation, in which all functionalities of pandas are described. - Ask the developers on Gitter. While primarily for development questions, the developers are happy to help users with other problems.

16.3.6   System Administrators

According to Rozanski and Woods, system administrators operate systems after deployment. We deem this category irrelevant to pandas, since it is a data analysis library. Hence, no stakeholders fall into this category.

16.3.7   Users

There are between five to ten million users of pandas world-wide [15], including major companies such as Uber, AQR, AppNexus and Datadog [16] [17], as well as many research institutes and data analysts.

---

[13]Nick Rozanski and Eóin Woods. 2012. Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives. *Addison-Wesley Professional.*

[14]Nick Rozanski and Eóin Woods. 2012. Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives. *Addison-Wesley Professional.*

[15]Marc Garcia. Towards pandas 1.0. https://www.youtube.com/watch?v=kUloTjPPgzU&t=2m20s. Accessed On: 25 February 2019.

[16]Nikhil Joshi and Isabel Geracioti (2017, October 31). Turbocharging Analytics at Uber with our Data Science Workbench. https://eng.uber.com/dsw/. Accessed On: 25 February 2019.

[17]pandas. Python Data Analysis Library. https://pandas.pydata.org/. Accessed On: 28 February 2019.

Apart from this, there are numerous libraries that integrate pandas functionality into their API. Examples of these libraries are: Google Cloud BigQuery [18], Apache Arrow [19], and matplotlib [20].

We distinguish two groups of users: the ordinary and the dependent users. The ordinary users are the majority: these users only install pandas and use it to analyze their data. On the other hand, the dependent users do not only install pandas, but also rely on pandas for their own projects, income or business value. An example of a dependent user is Dask, which internally uses pandas [21].

16.3.8 Competitors

There are several other tools available for data analysis with a similar feature set. These can be regarded as competitors of pandas. These competitors of pandas include [22] [23] [24]:
- MatLab - Microsoft Excel - R - SAS - Stata - SQL

16.3.9 Ecosystem

Around pandas exists a whole ecosystem of tools to store, analyze, or visualize data. These tools work together or complement each other. Sometimes they also depend or build upon each other. This ecosystem helps pandas to remain focused on its functionality [25]. Tools that fall into this ecosystem include [26] [27] [28] [29] [30]: - Apache Arrow - Apache Spark - Dask - Google BigQuery - matplotlib - Ray - seaborn - statmodels - Vaex

16.3.10 Power-Interest Grid

To gain insight in the influence of stakeholders surrounding the pandas projects, one can look at the power-interest diagram seen in Figure 2. Although core members hold the most power, due to their experience and role of integrator, the pandas team is very open to suggestions from the community as well. We expect enterprise users to have more interest than individual users, with equal power. However, companies can project more power by hiring enterprise contributors. The project's direction is also influenced by competing

---

[18]Google. Using BigQuery with Pandas. https://googleapis.github.io/google-cloud-python/latest/bigquery/usage/pandas.html. Accessed On: 25 February 2019.

[19]Apache. Pandas Integration. https://arrow.apache.org/docs/python/pandas.html. Accessed On: 25 February 2019.

[20]pandas. Plotting with matplotlib. https://pandas.pydata.org/pandas-docs/version/0.13/visualization.html. Accessed On: 25 February 2019.

[21]Dask. Dataframe documentation. http://docs.dask.org/en/latest/dataframe.html. Accessed On: 28 February 2019.

[22]Microsoft. Excel. https://products.office.com/en/excel. Accessed On: 25 February 2019.

[23]pandas. Comparison with other tools. https://pandas.pydata.org/pandas-docs/stable/getting_started/comparison/. Accessed On: 25 February 2019.

[24]MathWorks. Data Analysis. https://www.mathworks.com/solutions/data-analysis.html. Accessed On: 25 February 2019.

[25]pandas. Pandas Ecosystem. https://pandas-docs.github.io/pandas-docs-travis/ecosystem.html. Accessed On: 25 February 2019.

[26]Google. Using BigQuery with Pandas. https://googleapis.github.io/google-cloud-python/latest/bigquery/usage/pandas.html. Accessed On: 25 February 2019.

[27]Apache. Pandas Integration. https://arrow.apache.org/docs/python/pandas.html. Accessed On: 25 February 2019.

[28]pandas. Plotting with matplotlib. https://pandas.pydata.org/pandas-docs/version/0.13/visualization.html. Accessed On: 25 February 2019.

[29]pandas. Pandas Ecosystem. https://pandas-docs.github.io/pandas-docs-travis/ecosystem.html. Accessed On: 25 February 2019.

[30]Apache. PySpark Usage Guide for Pandas with Apache Arrow. https://spark.apache.org/docs/latest/sql-pyspark-pandas-with-arrow.html. Accessed On: 25 February 2019.

tools, because pandas strives to stay relevant. Similarly, the ecosystem surrounding pandas steers the responsibilities and focus of pandas' development.
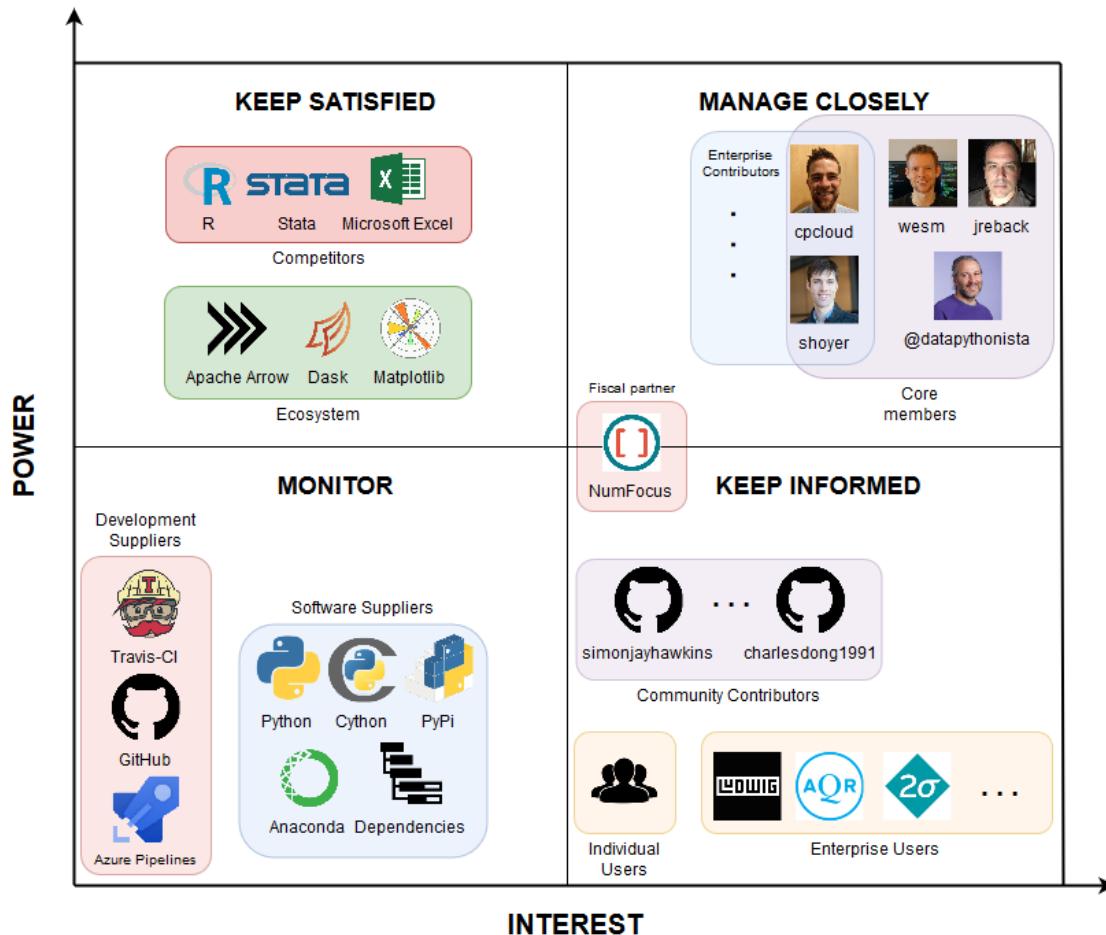
Figure 2: A power-interest grid according to our view of pandas.

16.3.11   Decision Making Analysis Results

We performed an extensive analysis of various information sources regarding the decision making process of the stakeholders, based on our pull request (PR) analysis in Appendix C.

The core team is the most prominent group of stakeholders in the PRs. Most discussion is about adding or revising tests (e.g. in PR 12, 14, 16, 19, 20), even for non-existent or not fully implemented features. There are also comments on API-breaking changes, which are often deferred to new issues (e.g. in PR 13, 15, 16). Moreover, sometimes there is discussion on functionality that diverges from standards and/or scientific conventions (e.g. in PR 13, 16). Rebasing can also trigger discussions, since sometimes tests break unexpectedly after a rebase. Finally, performance is also taken into consideration (see Performance Perspective).

Ordinarily, a PR gets merged if it complies with the contribution guide's requirements: whether the appropriate tests were written, old tests pass, documentation is provided and common standards are followed. Most of the time, the person that guards these aspects is core team member Jeff Reback, who seems to be omnipresent among the pull requests.

For the rejected PRs, it is often the case that the PR is abandoned, because either the PR is superseded by another one (e.g. in PR 1, 2, 7) or because the original author no longer works on it (e.g. in PR 8, 9). Furthermore, PRs can be rejected if they break backwards compatibility; changes to core packages are typically not tolerated.

16.4   Context View

To visualize the stakeholders, dependencies and competitors of pandas, we created the context view (see Figure 3). It describes the relationships between pandas and its environment.
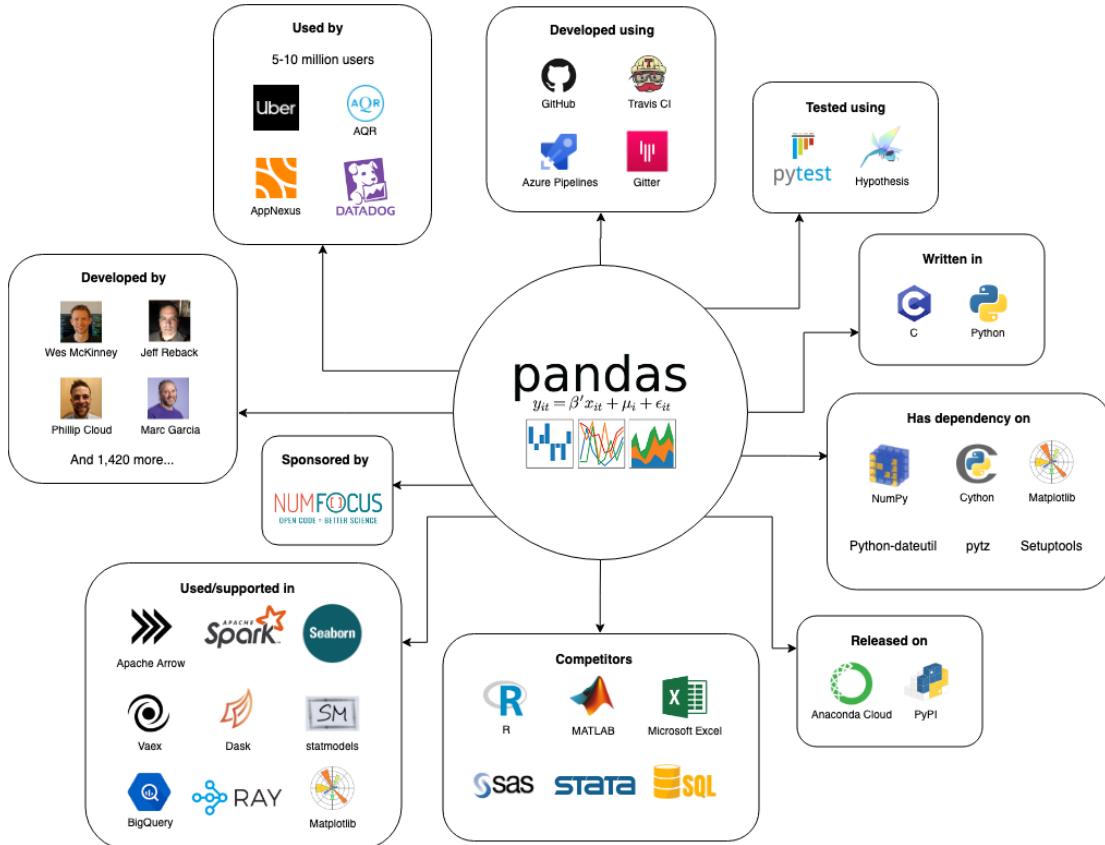
Figure 3: Context view of pandas.

Pandas is an open source project developed on GitHub written in Python, with some parts using C extensions

to increase performance for certain operations [31], for which Cython is used. Pandas is tested via pytest and Hypothesis, with Travis CI and Azure Pipelines for continuous integration. Communication between the developers is done via Gitter. Finally, pandas is released on Anaconda Cloud and PyPI.

Pandas is a data analysis tool that has several competitors with similar feature sets: R, MatLab, Microsoft Excel, SAS, Stata and SQL [32] [33] [34]. There are also other data analysis frameworks and tools that support or depend on pandas, some examples being: Apache Arrow, Apache Spark, seaborn, Vaex, Dask, statmodels, Google BigQuery, Ray and matplotlib [35] [36] [37] [38] [39]. Pandas itself has 45 dependencies [40], some noteworthy dependencies are: NumPy, Cython, Matplotlib, python-dateutil, pytz and Setuptools [41] [42].

Pandas is sponsored by NumFOCUS and is part of the PyData program [43], which is an educational program organized by NumFOCUS [44]. In Figure 4, we visualized the projects in the PyData program to show related data analysis tools.

---

[31]pandas. Contributing to pandas. https://pandas-docs.github.io/pandas-docs-travis/development/contributing.html. Accessed On: 25 February 2019.

[32]Microsoft. Excel. https://products.office.com/en/excel. Accessed On: 25 February 2019.

[33]pandas. Comparison with other tools. https://pandas.pydata.org/pandas-docs/stable/getting_started/comparison/. Accessed On: 25 February 2019.

[34]MathWorks. Data Analysis. https://www.mathworks.com/solutions/data-analysis.html. Accessed On: 25 February 2019.

[35]Google. Using BigQuery with Pandas. https://googleapis.github.io/google-cloud-python/latest/bigquery/usage/pandas.html. Accessed On: 25 February 2019.

[36]Apache. Pandas Integration. https://arrow.apache.org/docs/python/pandas.html. Accessed On: 25 February 2019.

[37]pandas. Plotting with matplotlib. https://pandas.pydata.org/pandas-docs/version/0.13/visualization.html. Accessed On: 25 February 2019.

[38]pandas. Pandas Ecosystem. https://pandas-docs.github.io/pandas-docs-travis/ecosystem.html. Accessed On: 25 February 2019.

[39]Apache. PySpark Usage Guide for Pandas with Apache Arrow. https://spark.apache.org/docs/latest/sql-pyspark-pandas-with-arrow.html. Accessed On: 25 February 2019.

[40]GitHub. Pandas Dependency Graph. https://github.com/pandas-dev/pandas/network/dependencies. Accessed On: 25 February 2019.

[41]pandas. Contributing to pandas. https://pandas-docs.github.io/pandas-docs-travis/development/contributing.html. Accessed On: 25 February 2019.

[42]pandas. Pandas Dependencies. https://pandas.pydata.org/pandas-docs/stable/install.html#dependencies. Accessed On: 25 February 2019.

[43]PyData. PyData Downloads. https://pydata.org/downloads/. Accessed On: 25 February 2019.

[44]PyData. PyData Mission. https://pydata.org/mission/. Accessed On: 25 February 2019.
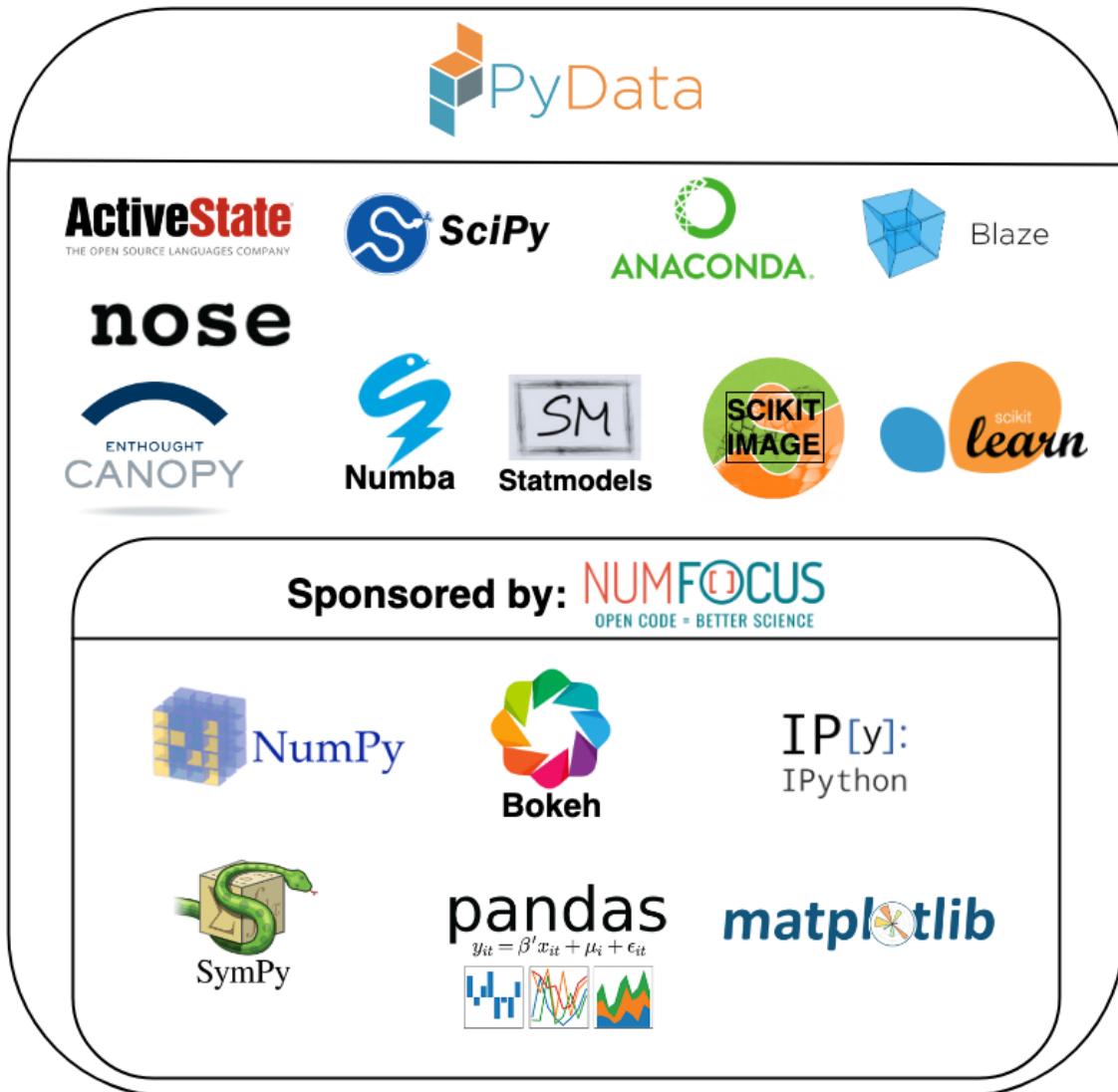
Figure 4: Data analytic tools in the PyData program.

Because pandas is rapidly growing in terms of popularity, its scope and responsibilities are under discussion. Originally pandas used NumPy at its core, however due to pandas' rapid growth, NumPy's pace of change was not fast enough anymore. The deciding factor was adding support for categorical data; this was not possible using NumPy without an excessive amount of effort. There was a discussion whether pandas should take over NumPy's role or whether they should directly work on NumPy themselves. In the end, pandas chose to implement the missing features and data types themselves. This means that pandas' scope is changing from purely an data analytics frameworks to the standard for data objects in the Python community.

16.5   Development View

To obtain insight into the architecture of the pandas repository, we focused on the Development Viewpoint described by Rozanski and Woods [45]. In this section, we describe the module organization, common processing, the standardization of both design and testing, and lastly the codeline organization. Notice that we do not cover the instrumentation concerns described by Rozanski and Woods, because pandas does not perform logging of runtime information, mainly due to pandas being a library instead of an application.

16.5.1   Module Organization

Within the pandas project, a distinction is made between the core modules and the rest of the project. The rest of the modules are either utilities for specific domains (IO, error messages, patterns, etc.), tests, compatibility features for older Python versions, or extensions of pandas (plotting, time series, etc.). These make up the outskirts of the pandas codebase. An overview of the codebase can be viewed in Figure 5. Evident is that the `io` and `core` packages contain the largest amount of code, and that the `io` and `_libs` packages consist mostly of C code. Notable is the relative size of the `io` package compared to the core package, indicating that pandas is quite invested in IO operations. Lastly, the `_libs` package consists of entirely C and Cython code. This package contains highly optimized code, used in other parts of the library.

In the `core` package, functionality is split up and grouped in subpackages. In Figure 5, this is evident from the subpackages in each package, with the `groupby` package as an example to split the grouping functionality from the rest of `core`. However, the splitting of modules is not universally enforced, evident from, for example, the DataFrame module `frame.py`, which contains a lot of different functionality. Members of the core development team have expressed an interest in trying to refactor some of these large modules, but have also mentioned that they do not know how to split them up properly due to all of the coupling between the features within them [46].

---

[45]Nick Rozanski and Eóin Woods. 2012. Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives. *Addison-Wesley Professional.*

[46]Joris van den Bossche. Reorganizing the megamodules. https://mail.python.org/pipermail/pandas-dev/2016-January/000448.html. Accessed On: 3 March 2019
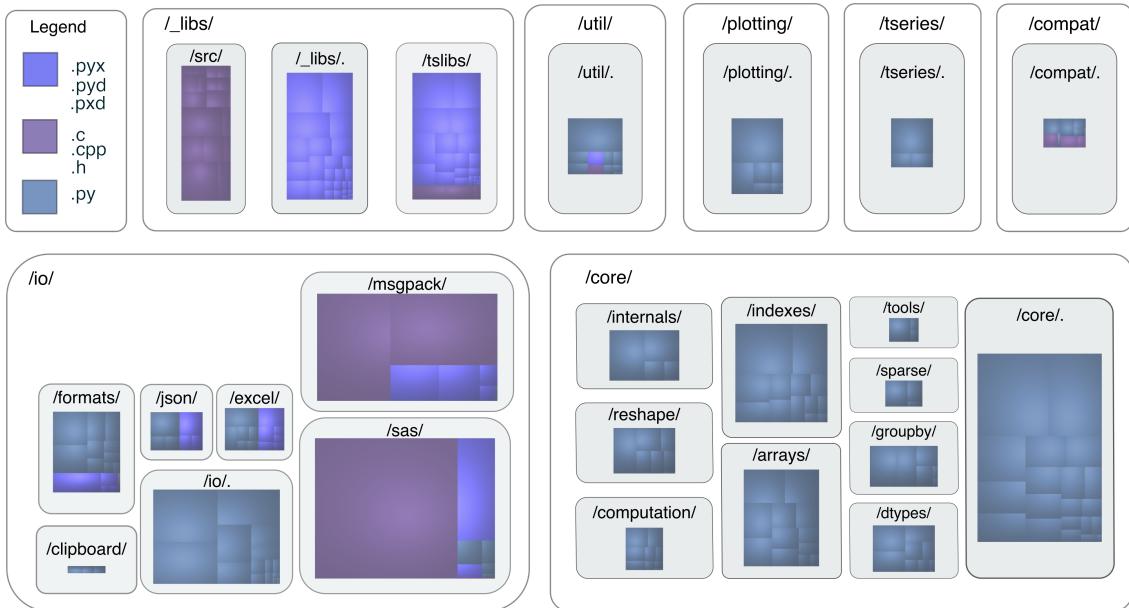
Figure 5: Overview of the file organization. The color indicates the file type, while the area of the rectangles represents the size of a source file.

Figure 6 shows a subset of the modules (those that are greater than 65Kb in size) along with the dependencies between these modules. Due to the sheer amount of dependencies and modules, not all dependencies could be visualized in readable manner. We focused on the relations in the `core` package. Even with this reduced visualization, it shows that there is a lot of tight coupling among pandas' modules, including quite a few circular dependencies (e.g. between `frame` and `series`).
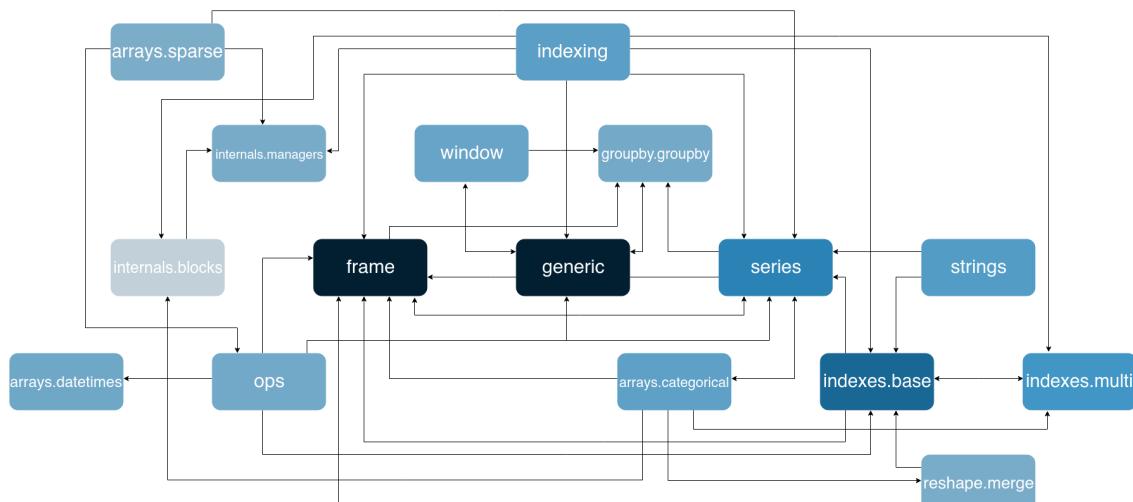
Figure 6: Dependency graph of pandas' largest `core` modules. The larger a module, the darker its color is. Only files which are greater than 65Kb in size are included to improve readability.

16.5.2   Common Processing

There are three areas in which common functionality is defined. First, there are the `common.py` modules present in some packages, for example in `core`. These modules include functions which are used throughout the code and which have generic applications, such as flattening data structures. Moreover, there is the `util` package, which provides functionality which enhances the developers' efficiency, such as decorators. Lastly, there is the `io` package, which is used to import and export from various data formats, like `csv` and `excel` files.

16.5.3   Standardization of Design

Pandas describes an extensive way of creating a consistent code base in its contribution guide [47]. This guide describes standards for its Python and C code, as well as for its documentation. Continuous Integration is used to enforce these standards.

Pandas does not explicitly document any design choices on a system basis. However, we did identify a recurring pattern in the code. There are multiple packages, which provide functionality to the main datatypes of pandas: Series and DataFrame. This means that the functionality of both these datatypes is abstracted to another package. An example of this can be found in the `groupby` package, which provides grouping functionality for data in DataFrames.

16.5.4   Standardization of Testing

The testing framework used in pandas is pytest. The contribution guide encourages the use of Test Driven Development, showing that pandas takes testing seriously. Although the guide does state that the test modules should be placed inside of the testing subdirectory of the package under testing (e.g. the tests for `core/frame.py` should be under `tests/core/`), there are no other explicitly stated styling standards enforced on the tests; the guide encourages to look at other tests for inspiration. However, there is a specific testing page on the wiki [48], which describes the methodology to accomplish certain testing objectives, such as testing methods using files, or network connection.

Similarly to the coding standards, the tests are ran by CI as well for each contribution, failing the build whenever a test fails. Moreover, pandas utilizes Codecov to view the difference in line coverage between the contribution and the master branch. This statistic is used as well to determine whether more tests have to be added to the contribution.

---

[47]pandas. Contributing to pandas. https://pandas-docs.github.io/pandas-docs-travis/development/contributing.html. Accessed On: 25 February 2019.

[48]pandas. Testing. https://github.com/pandas-dev/pandas/wiki/Testing. Accessed On: 07 March 2019.

16.5.5   Codeline Organization
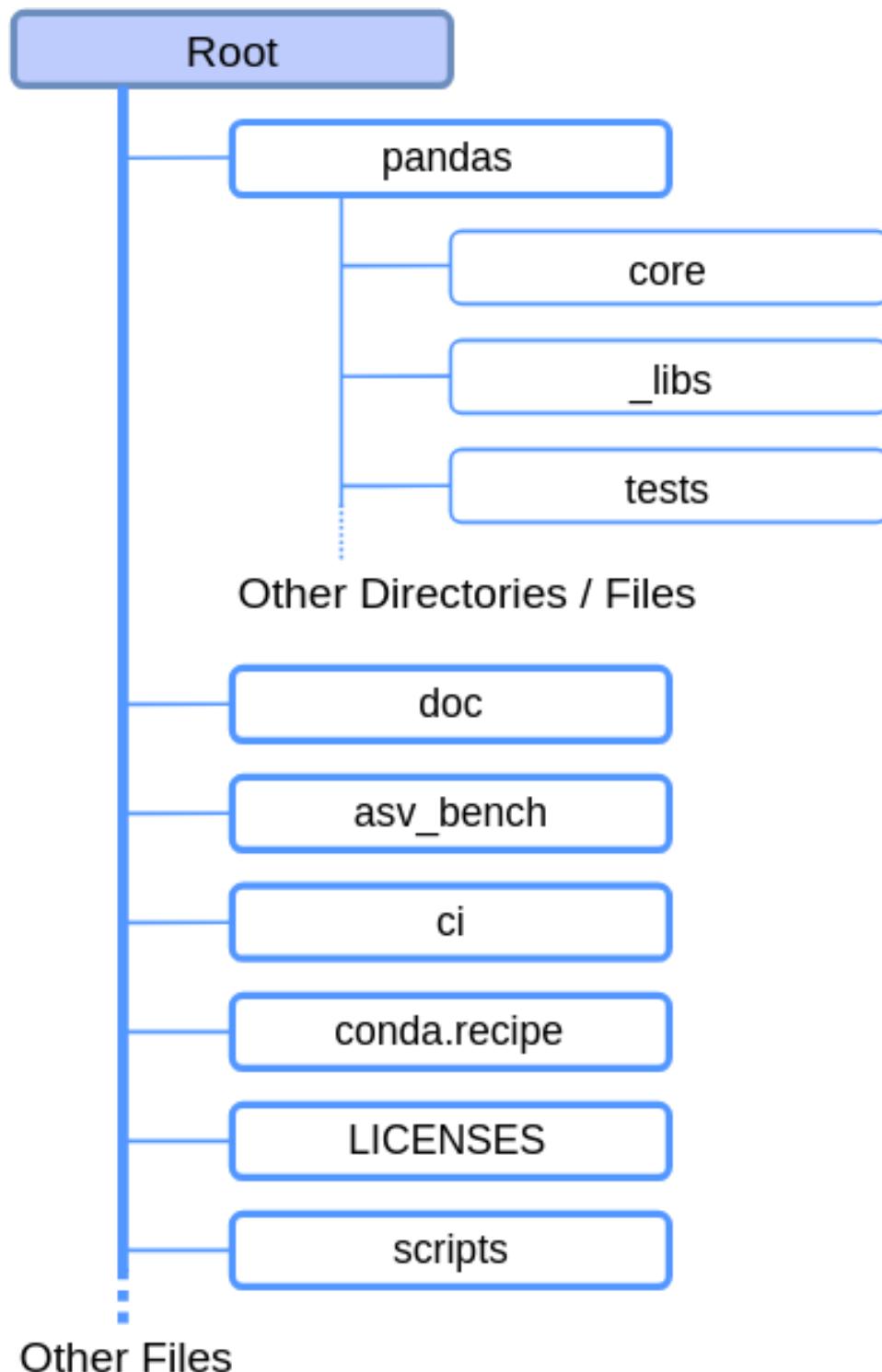
Figure 7: Directory structure of the pandas repository. The dotted lines indicate one or more subdirectories or files

being ommitted from the image.

The pandas project includes more than just the source modules described in the module organization section. Figure 7 shows an overview of the complete project, which include the following directories: - The `doc` directory contains all assets for building the documentation. In this directory, many `reStructuredText` files are present, which are used for building the documentation using Sphinx. - The `asv_bench` directory provides configuration for the benchmarking software Airspeed Velocity (asv). - The `ci` folder includes all configuration for both Azure Pipelines and Travis CI, to perform static analysis on the code for pull requests. - The `conda.recipe` directory contains build scripts for Anaconda. - `LICENSES` contains all licenses of third party software used in the development of pandas. - The `scripts` directory includes scripts which are used to simplify processes that are specific to contributors, such as testing or static analysis.

There are also some miscellaneous files in the root directory of the pandas project, including files related to Git and GitHub.

16.6 Performance Perspective

For a library that deals with very large data sets on a regular basis, performance is a crucial aspect. For pandas, delivering good performance is therefore one of the key goals, allowing data analysts to effectively analyze very large data sets. Most of pandas is written in Python, but certain operations are sped up using C (mostly using Cython) [49], which allows for performance improvements that decrease the execution time up to 200 times [50]. Some other parts use Numba, which is a just-in-time compiler that translates Python and NumPy code to native machine instructions, leading to C-like performance [51] [52]. Numba is best applied to methods that apply numerical functions to NumPy arrays. Furthermore, for very large data sets, pandas has an `eval()` expression evaluation method, which in certain scenarios can give an ~2 times performance increase, since it provides out-of-core computation and can be run in parallel [53].

The main performance concern of pandas is how long certain operations take, to create acceptable performance levels for large data sets. Pandas uses the Airspeed Velocity (asv) benchmark to compare the performance of their codebase with previous versions. This benchmark contains a plot for each method showing how the speed evolved over each iteration of pandas. The asv benchmark is ran continuously (can be found here) and is used to determine, in a systematic way, whether the performance of pandas did not regress and if it did, it shows what functions are affected.

The performance requirement of pandas is not well defined, but it should perform similar to other data analysis tools and frameworks, enabling data scientists to work with large data sets. Therefore, the performance of pandas is often compared to those similar tools and frameworks by third parties (for example Fast-Pandas and db-benchmark). On the pandas GitHub repository, users can submit their performance issues, which will then be labelled with the `Performance` label (As of 11 March 2019, 5.5% of all of pandas' issues are labelled with the `Performance` label). The core development team actively takes part in discussing

---

[49]pandas. Contributing to pandas. https://pandas-docs.github.io/pandas-docs-travis/development/contributing.html. Accessed On: 25 February 2019.

[50]pandas. Enhancing Performance. https://pandas.pydata.org/pandas-docs/stable/user_guide/enhancingperf.html. Accessed On: 7 March 2019.

[51]pandas. Enhancing Performance. https://pandas.pydata.org/pandas-docs/stable/user_guide/enhancingperf.html. Accessed On: 7 March 2019.
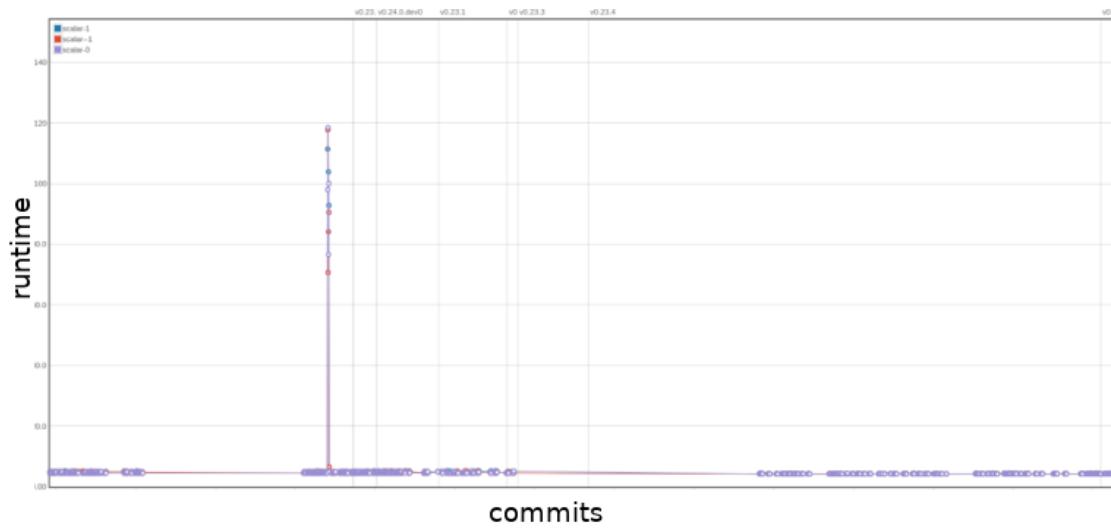
[52]Numba. Numba. http://numba.pydata.org/. Accessed On: 7 March 2019.

[53]pandas. Contributing to pandas. https://pandas-docs.github.io/pandas-docs-travis/development/contributing.html. Accessed On: 25 February 2019.

these issues, asking the developers, that are working on the issue, to run the asv benchmark locally and post the results in the PR [54]. If the methods in question are not yet included in the asv benchmark, they will also ask the developers to update the benchmark to be able to automatically measure the performance/regression of the methods in the future [55].

16.6.1   Analysis of Airspeed Velocity Benchmark

In the asv benchmark we can see that sudden extreme changes in runtime are often fixed swiftly, resulting in peaks in the plots. One example of this can be seen in Figure 8 of the `add-overflow-scalar` function under the `binary_ops` section. We also noticed that regression in future versions was minimal if an effort was made to speed up a method. However, the methods that went without a speed up, often have one large performance decrease, after which their performance remains constant. One example of this can be seen in Figure 9, which shows the benchmark of the `add_overflow_arr_mask_nan` function under the `binary_ops` section. Most of the time, the performance remains constant. If however, a performance change occurs, it is rarely gradual, instead the performance will often jump drastically from one version to the next.

Figure 8: Benchmark of `add-overflow-scalar` under the `binary_ops` section. The x-axis are commits in chronological order, the y-axis is the runtime of the function. The up to date benchmark can be found here.

---

[54]Adam Merberg. pandas.Series.isin() is slow on large sets due to conversion of set to list. https://github.com/pandas-dev/pandas/issues/25507. Accessed On: 11 March 2019.

[55]Ilya Silchenkov. Slowdown of str(record) of DataFrame.to_records() method. https://github.com/pandas-dev/pandas/issues/21880. Accessed On: 11 March 2019.
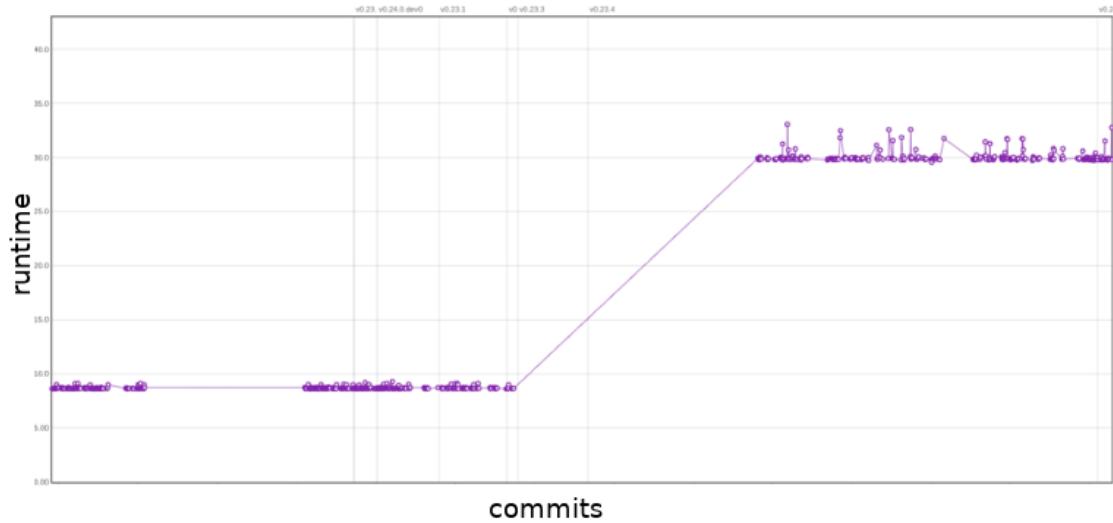
Figure 9: Benchmark of `add_overflow_arr_mask_nan` under the `binary_ops` section. The x-axis are commits in chronological order, the y-axis is the runtime of the function. The up to date benchmark can be found here.

An example of how the asv benchmark is used can be seen in issue #18532. The core development team noticed a regression in the `Series` constructor. In a follow-up PR this was fixed and that performance has been maintained since then. This increase in performance can be observed in Figure 10, where we see a sharp drop in runtime from 369ms to 155µs.
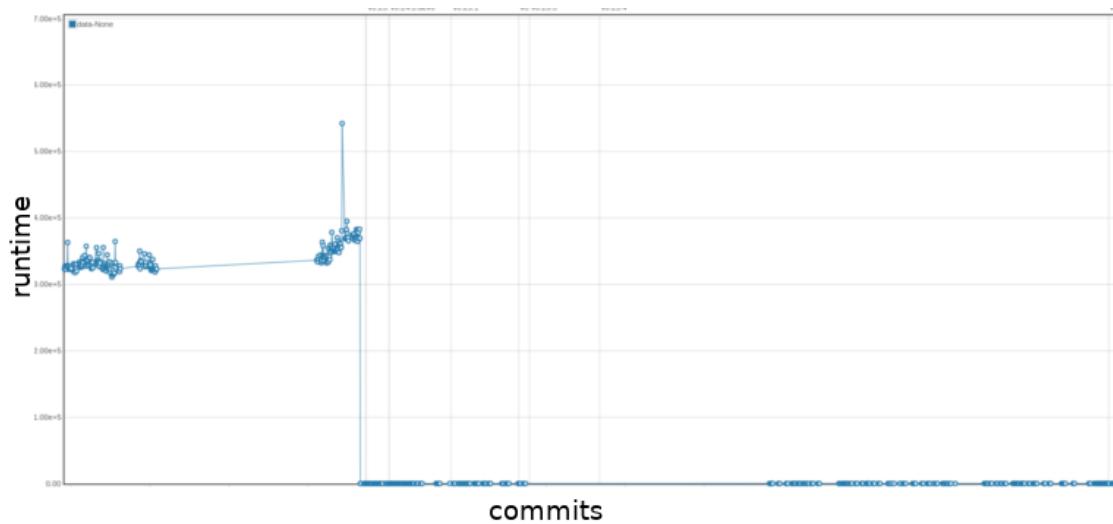
Figure 10: Benchmark of the `Series` constructor method. The x-axis are commits in chronological order, the y-axis is the runtime of the function. The up to date benchmark can be found here.

16.7   Technical Debt

In this section, we analyze the technical debt of the pandas project. First we analyze the code and testing debt of pandas. Lastly, we investigate how the developers discuss the technical debt.

16.7.1   Automatic Analysis

SonarQube was used to assess the technical debt in pandas. SonarQube is able to determine the amount of bugs, vulnerabilities, technical debt, code smells and the amount of duplication. Unfortunately SonarQube was not able to automatically provide the code coverage, but for this, other tools were used (see the Testing Debt section). As can be seen in Figure 11, pandas has 70 days worth of technical debt, but still gets an A rating. Projects will receive an A rating when the technical debt is less than 5% of the total code base. Since pandas itself is 256k lines of code (according to SonarQube), this seems reasonable.
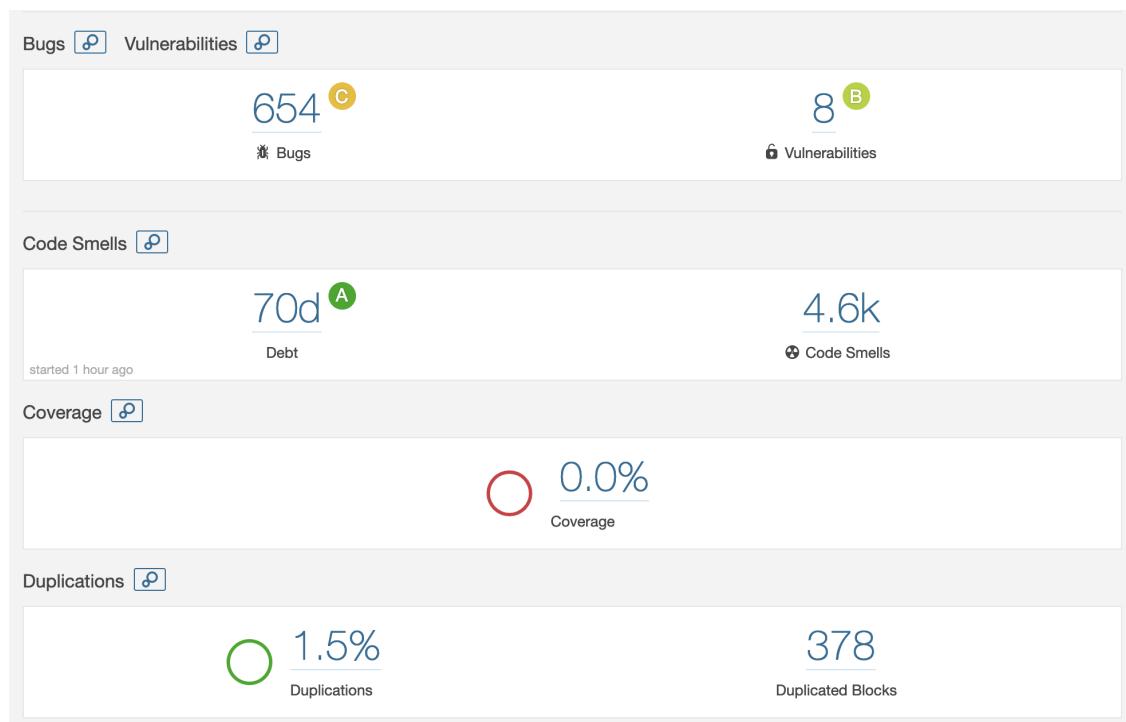
Figure 11: SonarQube's dashboard for the pandas project.

SonarQube found 4.6k code smells, of which 2.4k are about deprecated HTML attributes estimated to be 25 days of effort. Upon further inspection, these HTML attribute smells are all located in the test directory, verifying whether pandas can read from these HTML pages. This means that these are false positives.

The other most found code smells are (in decreasing order of occurrences): high cyclomatic complexity (even as high as 97), unconventional parameter and function names (often one letter names), commented-out code (often part of reminders of bugs or TODO's) and functions with too many parameters. The functions that have too many parameters are mostly in the user-facing API methods (e.g. `to_csv`). Most of the

parameters are initialized with a default value, but can be configured by the user to fit his/her purpose more. Thus we find these to be false positives as well.

Of the 654 bugs found by SonarQube, 499 are deprecated HTML elements and tags, in the same classes as before in the code smells. Most of the remaining bugs are that of identical sub-expressions on both sides of (in)equality operators (134 occurrences). These are all found in the test methods, to test the overrides of the equality methods. Thus at least 633 of the found bugs are false positives.

16.7.2 Manual Analysis

To further quantify the state of technical debt in pandas, we analyzed to what degree pandas adheres to the SOLID principles.

16.7.2.1 Single Responsibility Principle (SRP)

We consider responsibilities on a module basis in addition to just classes. In many modules, multiple class definitions were found. Table 1 shows the largest 10 modules in terms of method definitions. The largest module appears to be `pytables.py`, with 272 definitions among 33 classes that are abstractions of different types of tables and utility objects. In this case, each table type does have a clear responsibility, but the module does not. For example, `generic.py` contains 220 function definitions stemming from a single class, clearly carrying many responsibilities.

Table 1: 10 largest modules of pandas in terms of function definitions.

Module name	Number of function definitions
`pytables.py`	272
`generic.py`	232
`base.py`	216
`blocks.py`	185
`frame.py`	151
`_core_.py`	144
`offsets.py`	139
`series.py`	131
`window.py`	125
`managers.py`	125

16.7.2.2 Open-Closed Principle (OCP)

For the OCP we look at how extendible the code is. In general, extending in pandas is made easier by allowing users to decorate existing packages without sub-classing them [56]. This can be convenient, but can break the open-closed principle. Pandas has several occurrences of OCP violations, for example private variables that are not exposed for extension (e.g. `core/computation/ops.py`). Lastly, most of the optimized Cython code is difficult to cleanly extend.

16.7.2.3 Liskov Substitution Principle (LSP)

In terms of LSP, it is hard to find potential violations in pandas, since it rarely occurs that subclasses in inheritance relations override existing behavior. Especially since many super-classes in pandas only have one child. Python is also not statically typed, hence looking for type casting is not a useful heuristic.

---

[56]pandas. Extending Pandas. https://pandas.pydata.org/pandas-docs/stable/development/extending.html. Accessed on 21 March 2019.

We observed one instance were behavior overriding did occur: the `pandas.io.sql` module. For each of the overridden methods in `SQLiteTable` the pre-conditions were equal and the post-conditions were at least as strong than its super-class. This means that pandas seems to comply to the LSP for this instance.

16.7.2.4 Interface Segregation Principle (ISP)

For the interface-segregation principle we checked the amount and relevance of functions provided when importing a package. In Python, ISP can be tricky, as it does not cleanly allow interfacing of code, so implementations are used directly.
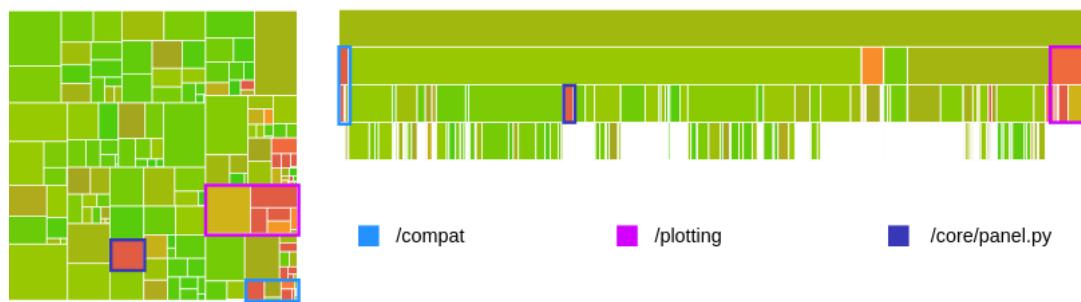
A violation can be found by importing pandas: `import pandas as pd`. Over 50 functions, fields and casses are returned when inspecting `pd`. One would expect the `pd` object to contain purely user-side code, but there are various functions that do not directly relate to each other and are often irrelevant to the user.

16.7.2.5 Dependency Inversion Principle (DIP)

Lastly, from the developer's view we consider core code depending on non-core code to be a DIP violation. From the user-side, it is a violation whenever a user can reach core code. A quick inspection using `import pandas as pd` reveals that core modules are abstracted away neatly; they cannot be reached. However, we see multiple occurrences of core modules importing and tightly coupling with non-core modules (e.g. `pytables.py` importing `io` modules).

16.7.3 Testing Debt

As mentioned in the standardization of testing section, pandas takes testing seriously. Codecov is used to analyze the current line coverage of Python code within the repository. The C code is not taken into consideration for the analysis since it is generated through Cython. In Appendix D, the coverage for the master branch is presented, showing that 91.2% of the 52.977 lines are covered. Some visualizations from Codecov are displayed in Figure 12 below.

Figure 12: Two code coverages perspectives generated by Codecov. The left picture is a grid overview, with rectangles being files. On the right, a view is given in the form of sub-directory layers. The size of a rectangle represents the size of the file and the color represents the number of tested lines.

Using both Appendix D and Figure 12, we identify three packages and modules of interest: the `core/panel.py` module, as well as the `compat` and `plotting` packages.

The pandas team is currently in the process of completely removing `core/panel.py`, which has been deprecated since pandas version 0.20 [57]. Hence, no further maintenance is being performed on the module, explaining the low coverage.

Two other packages that are not well tested are `compat` (for NumPy compatibility) and `plotting` (for matplotlib support), with 63.32% and 72.72% line coverage respectively. The main part of uncovered code in the `compat` package is that of initialization with Python 2.7. Since pandas is dropping support for Python 2.7 in 2020 and all new feature releases will be in Python 3 only [58], this code is soon to be deprecated and removed. In the `plotting` package however, code in `_converter.py` is largely untested. This likely stems from dealing with the `matplotlib` integration, which complicates the process.

Overall, we can say that pandas tests their Python code quite well, with the main low-coverage offenders being deprecated code.

16.7.4 Historical Analysis

We analyzed the technical debt using SonarQube for all main releases of pandas and created the plots seen in Figure 13. In these plots, we ignored the technical debt introduced by the HTML files, as they are false positives and correspond to more than half of the total amount of technical debt (as discussed in Automatic Analysis). We can see that the amount of code, technical debt and code smells increases linearly over the different releases. We can also see that the newest main release (v0.24.0) brings a decrease in the technical debt and code smells, while still introducing more lines of code.
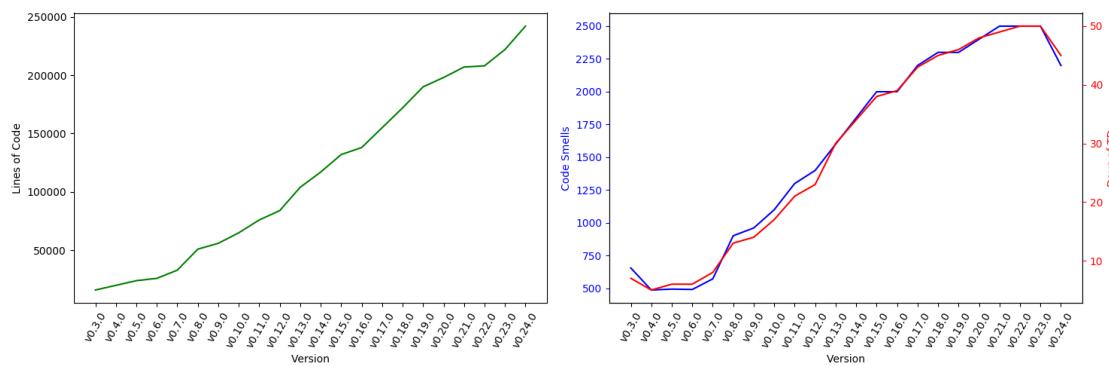
Figure 13: History of technical debt of the pandas project, excluding the HTML files.

16.7.5 Technical Debt Discussions

Technical debt is an important subject for discussion in the pandas team. A lot of insights in how technical debt is discussed can be gained from the pandas mailing list and from the way the developers use constructs such as FIXMEs and TODOs.

---

[57] pandas. pandas.Panel. https://pandas.pydata.org/pandas-docs/version/0.23.4/generated/pandas.Panel.html. Accessed On: 18 March 2019.

[58] pandas. Plan for dropping Python 2.7. https://pandas.pydata.org/pandas-docs/stable/install.html#plan-for-dropping-python-2-7. Accessed On 18 March 2019.

At the end of 2015, pandas' fast growth was a key reason for the core team to start discussing major refactorings via their mailing list. Wes McKinney was pushing for large internal refactorings to expose less of the core to the end-users [59]. It was mentioned that some features, required a large amount of effort due to the architectural issues with pandas and its over-reliance on NumPy's ndarray data format, which did not properly support the new needs of pandas (e.g. categorical data). Later on, there was discussion about releasing pandas 1.0, so that the focus could shift to bug-fixing. The idea was to start a new pandas 2.0 development branch for the major refactorings [60]. However, Joris van den Bossche indicated that this may hinder contributors, since new feature PRs would then not be welcome on either branches. Pandas 2.0 is still actively discussed and pandas' creator Wes McKinney has shifted his attention to Apache Arrow, which is intended to be at the core of pandas 2.0 [61] [62] [63]. The mailing list also contains lots of discussions about smaller technical debt issues, an example being a violation of the DRY principle, where both Series and DataFrame had a subclass for sparse data [64].

The pandas developers also make use of FIXMEs and TODOs to discuss about technical debt. We analyzed the FIXME and TODOs at the repository's state from commit 79205ea. It became apparent that FIXMEs and TODOs are rarely linked to GitHub issues, making it hard to determine whether they were already taken care of. Most of the FIXME comments are in test modules, which seems to indicate that the test modules are suffering from technical debt. TODOs are more present in non-test modules. `ops.py` was one of the non-test modules that had GitHub issues and PRs associated with its FIXMEs (5284, 5035, 19448). Noteworthy is that the linked PR is not really related to the described issue, but rather where it came up in.

16.7.6   Before-After Analysis Of Technical Debt Issue

We decided to contribute to pandas by solving an SRP violation that we identified [65]. Before our refactor, there was a single test module for all the sparse series tests. This clearly violated the SRP principle, since it took responsibility over all the features of sparse series.

The refactor involved categorizing the individual tests of the original module, so that they could be moved to smaller and more specific modules. These are then responsible for testing specific types of functionality, much like how the regular series tests are split up into constructor, api, missing, indexing, and other types of tests.

After the refactor, this module was split up into smaller modules, hence eliminating the SRP violation, since all of the smaller modules now have a single responsibility.

---

[59] pandas mailinglist. January 2016 entry. https://mail.python.org/pipermail/pandas-dev/2016-January/thread.html. Accessed on 18 March 2019.

[60] pandas mailinglist. July 2016 entry. https://mail.python.org/pipermail/pandas-dev/2016-July/thread.html. Accessed on 18 March 2019.

[61] pandas Development Team. Pandas 2.0 Design Documents. https://pandas-dev.github.io/pandas2/. Accessed on 4 April 2019.

[62] Wes McKinney. Apache Arrow and the 10 Things I Hate About pandas. http://wesmckinney.com/blog/apache-arrow-pandas-internals/. Accessed on 4 April 2019.

[63] Tobias Macey (Host). Wes McKinney's Career In Python For Data Analysis - Episode 203. https://www.pythonpodcast.com/wes-mckinney-python-for-data-analysis-episode-203/ Accessed on 4 April 2019.

[64] pandas mailinglist. November 2018 entry. https://mail.python.org/pipermail/pandas-dev/2018-November/thread.html. Accessed on 18 March 2019.

[65] Jeff Reback. TST: split out sparse tests. https://github.com/pandas-dev/pandas/issues/18969. Accessed On: 8 April 2019

16.8   Conclusion

Pandas is an ever growing library for data analysis and is becoming the new standard in the Python community. We have analyzed pandas from different perspectives and viewpoints and found that pandas is awaiting large refactorings. Although pandas does not have a large amount of technical debt, its rapid growth is becoming a problem as its dependencies can not keep up with the growth. Due to this, the responsibilities of pandas are increasing, requiring a philosophical and architectural shift, that will ultimately benefit the usability of the stack. Pandas' creator Wes Mckinney has shifted his attention to Apache Arrow, which should unify the management of in-memory data representations across different toolkits and languages, which pandas 2.0 will also rely on. All in all, pandas is here to stay and is only becoming larger.

16.9   Appendix A: Core Team

Table 2: The complete core team as documented by the pandas governance repository [66].

Full Name	GitHub Handle
Andy Hayden	[@hayd](https://github.com/hayd)
Brock Mendel	[@jbrockmendel](https://github.com/jbrockmendel)
Chang She	[@changhiskhan](https://github.com/changhiskhan)
Chris Bartak	[@chris-b1](https://github.com/chris-b1)
G. Young	[@gfyoung](https://github.com/gfyoung)
Jeff Reback	[@jreback](https://github.com/jreback)
Jeremy Schendel	[@jschendel](https://github.com/jschendel)
Joris van den Bossche	[@jorisvandenbossche](https://github.com/jorisvandenbossche)
Marc Garcia	[@datapythonista](https://github.com/datapythonista)
Masaaki Horikoshi	[@sinhrks](https://github.com/sinhrks)
Matthew Roeschke	[@mroeschke](https://github.com/mroeschke)
Phillip Cloud	[@cpcloud](https://github.com/cpcloud)
Pietro Battiston	[@toobaz](https://github.com/toobaz)
Stephan Hoyer	[@shoyer](https://github.com/shoyer)
Tom Augspurger	[@tomaugspurger](https://github.com/tomaugspurger)
Wes McKinney	[@wesm](https://github.com/wesm)
William Ayd	[@willayd](https://github.com/willayd)

16.10   Appendix B: Suppliers

Table 3: Suppliers of the pandas library

---

[66]pandas. pandas People. https://github.com/pandas-dev/pandas-governance/blob/master/people.md. Accessed On: 25 February 2019.

Supplier	Type	Role
Python	Software	Language in which pandas is written
C/Cython	Software	Language in which pandas is written, and a program to bundle this with the Python code
PyPI	Software	Package manager used to retrieve dependencies
Dependencies	Software	Libraries from which functionality is used by pandas
GitHub	Development	Hosts the repository where pandas is developed
Gitter	Development	Communication platform of pandas developers
Azure Pipelines	Development	Continuous Integration used to analyze the code statically
TravisCI	Development	Continuous Integration used to analyze the code statically.

16.11   Appendix C: Decision making analysis

In order to analyze the decision making process, we analyzed 10 rejected and 10 accepted pull requests, along with consulting other sources such as mailing lists, conference slides, the contribution guide, gitter, and other miscellaneous sources.

16.11.1   Pull Request Analysis Methodology

To analyze the pull-requests, we combined the recommended way of doing it with some automation. We went into the pull request and collected the following contextual information: author, timespan, what the PR touches, reason for closing (if applicable), preceding issues, reviewers and finally, interesting notes. This mirrors what was suggested for collecting contextual information.

Then, instead of tagging the PRs manually, we automated the process using pandas itself together with Matplotlib. To tag the data, we first looked through some PRs manually to determine good tags to categorize the comments with. From this, we found trends that kept re-occuring and were easy to identify: git-related discussion, naming conventions, testing, and documentation. Anything that would not fit these categories, would be considered as feature-specific or other discussion.

Then we found some words that were frequently mentioned in one form or another for each of these categories, e.g. for the git tag we used the following list of regex patterns:

```
"git" : [r"squash(ed)?",r"rebas(e)?", r"\bgit\b", "push", r"merge( conflict)?",
r"pull( request)?", "commit"]
```

Similar lists exist for the other categories. Upon review, the categorization yielded good results, with relatively few false positives. To better follow conversations, we added another tweak: if the algorithm

could not find a new specific tag, it would use the tag of the previous comment instead of labelling it as feature/other. This yielded much better results than before, with a slight increase in false positives. Another addition that improved results was always re-checking the tag when entering a review comment chain. Most of this could be done because of temporal dependency between comments: related comments often are grouped together closely in time [67]. The system could be improved even further if it would consider users being tagged in replies (for replies to comments that are far apart from the original in terms of time).

Overall, the tagging yields a swarm plot, which visualizes the density of activity at various time periods for each of the tags, a pie chart that shows the distribution of tags among the comments, and a sequence plot that visualizes the ordering of comment tags in time more clearly. Aside from that, an additional pie chart of the user activity is also made, where any user with less than 2% of the comments is grouped under the category "other".

16.11.2 Most Discussed Rejected Pull Requests
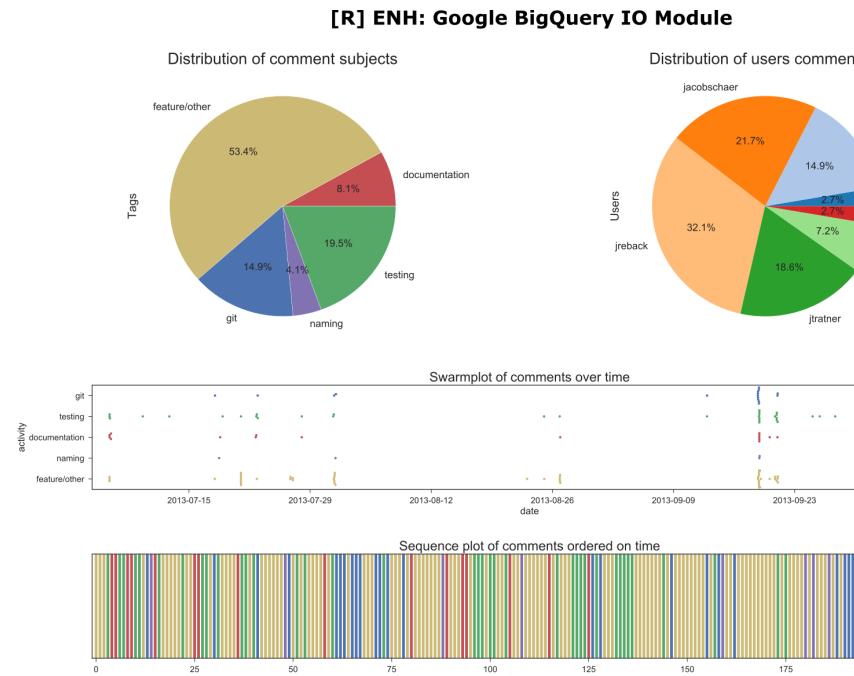
16.11.2.1 1. ENH: Google BigQuery IO Module #4140

16.11.2.1.1 Context

- Author: sean-schaefer
- Timespan: 2013-07 : 2013-10
- Touches: Integration of the pandas.io.gbq module for integration with Google's BigQuery
- Reason closed: Superseded by other pull request: #5179.
- Preceding issues: N/A
- Reviewers: cpcloud (main), jreback (comments), jratner
- Notes:
    - This PR required some communication between Google's backend team and the pandas contributors.
    - There was a significant discussion after closing the PR about a continuation.
    - This PR required some communication between Google's backend team and the pandas contributors because they found a bug in Google BigQuery.

---

[67] Noé Gaumont, Tiphaine Viard, Raphaël Fournier-S'Niehotta, Qinna Wang, and Matthieu Latapy. Analysis of the temporal and structural features of threads in a mailing-list. InComplex Networks VII 2016 (pp. 107-118). *Springer, Cham.*

[R] ENH: Google BigQuery IO Module

Distribution of comment subjects

Distribution of users commen

Swarmplot of comments over time

Sequence plot of comments ordered on time
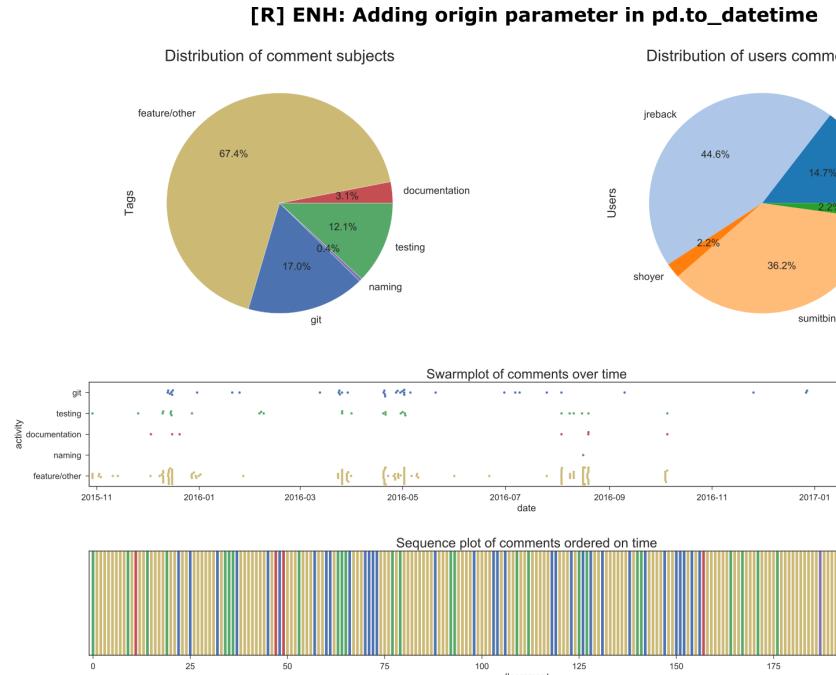
16.11.2.1.2 Summarizing Visualizations

Most of the activity on this PR seemed to be happen two days after its creation, with quite a bit of discussion on testing and other non-feature related subjects.

16.11.2.2   2. ENH: Adding origin parameter in pd.to_datetime #11470

16.11.2.2.1   Context

- Author: sumitbinnani
- Timespan: 2015-10 : 2017-03
- Touches: to_datetime method. Concerned adding a parameter to the pandas datetime module that allowed the user to set a time-origin
- Reason closed: Superseeded by other pull request: #15828
- Preceding issues: #11276, #11745
- Reviewers: jreback (main), shoyer, jorisvandenbossche.
- Notes:
    - An important request for this feature was adding the origin of the date-time as a parameter.
    - jreback spent quite some time guiding sumitbinnani in polishing his work.

[R] ENH: Adding origin parameter in pd.to_datetime

Distribution of comment subjects

Distribution of users commen

Swarmplot of comments over time

Sequence plot of comments ordered on time
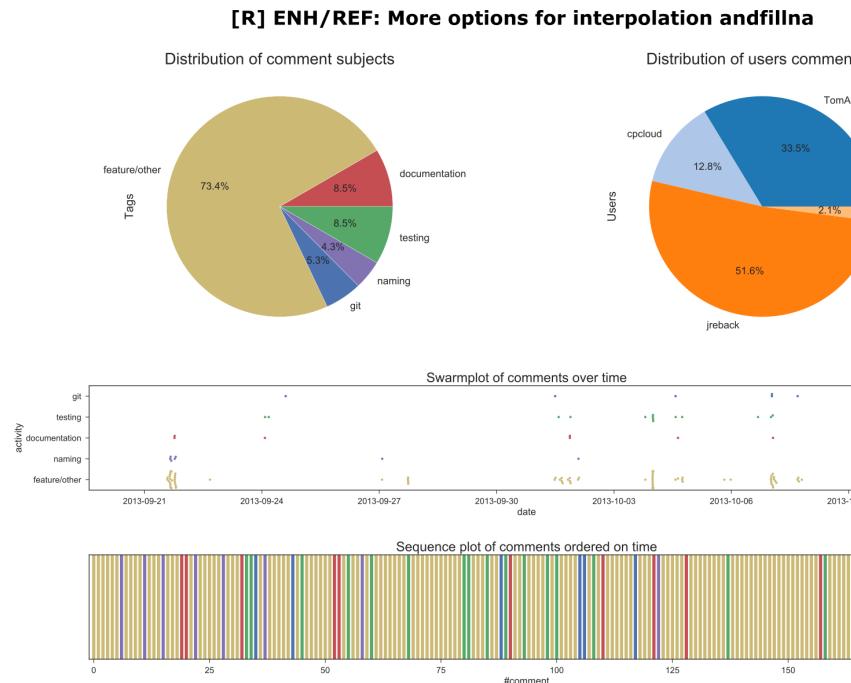
16.11.2.2.2 Summarizing Visualizations

The main discussion here was clearly between `jreback` and `sumitbinnani`. This PR was also open for quite a long time (nearly 1.5 years) with some bursts of discussions here and there.

16.11.2.3   3. ENH/REF: More options for interpolation and fillna #4915

16.11.2.3.1   Context

- Author: tomaugspurger
- Timespan: 2013-09 : 2013-10
- Touches: interpolation and fillna, adding more options.
- Reason closed: Pull request had to be squashed and a new pull request was made for this addition. Only a link to the merge commit was found.
- Preceding issues: #1892, #4434
- Reviewers: jreback (main), cpcloud, jratner.
- Notes:
    - The review process shows `jreback` guiding `TomAugspurger` intensively to work out his contribution.
    - The enhancement was appreciated, but had to be squashed and a new PR was made

[R] ENH/REF: More options for interpolation andfillna

Distribution of comment subjects

Distribution of users commen

Swarmplot of comments over time

Sequence plot of comments ordered on time
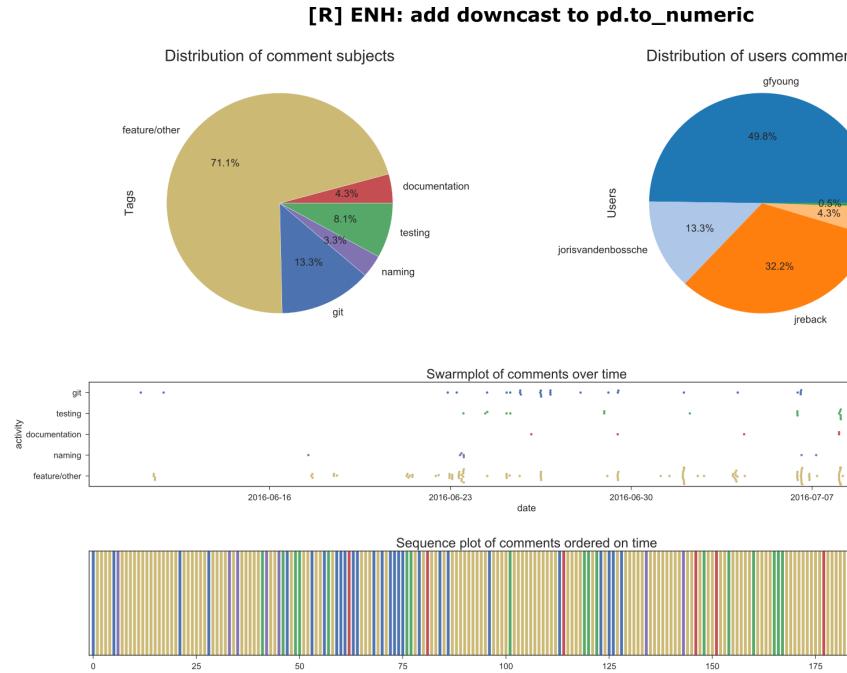
16.11.2.3.2 Summarizing Visualizations

As established, the main discussion here was between `jreback` and `TomAugspurger`. Quite a bit of comments on documentation towards the end.

16.11.2.4   4. ENH: add downcast to pd.to_numeric #13425

16.11.2.4.1   Context

- Author: gfyoung
- Timespan: 2016-06 : 2016-07
- Touches: to_numeric method. Concerned a change to the API for reading CSV's. Two functions had to be moved outside of the parser.
- Reason closed: Contribution was included in another pull request.
- Preceding issues: #13352
- Reviewers: jreback (main), jorisvandenbossche
- Notes:
    - The API changes generated quite some discussion.

[R] ENH: add downcast to pd.to_numeric

Distribution of comment subjects

Distribution of users commen
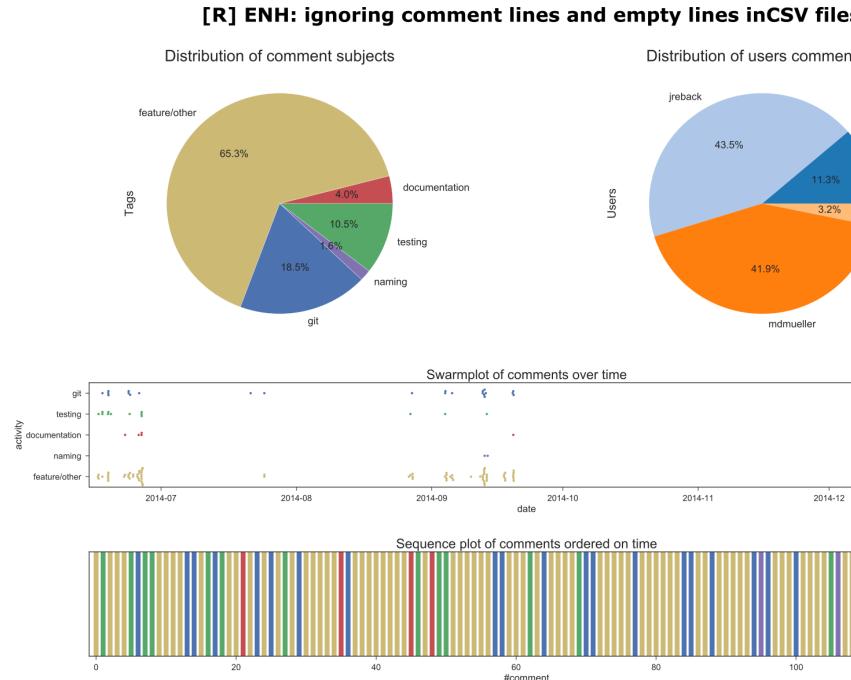
16.11.2.4.2 Summarizing Visualizations

The main discussion here is between `gfyoung` and `jreback`. There was quite a bit of discussion about git and CI at the beginning of the PR, which happened to be because of failing Travis builds.

16.11.2.5   5. ENH: ignoring comment lines and empty lines in CSV files #7470

16.11.2.5.1   Context

- Author: mdmueller
- Timespan: 2014-06 : 2014-09
- Touches: CSV reader. Concerned an enhancement to the CSV reader to ignore empty lines.
- Reason closed: Pull request was squashed into one commit and merged through that commit instead of the pull request.
- Preceding issues: #4466
- Reviewer: jreback (main), jorisvandenbossche.
- Notes:
  - A large chunk of discussion between `jreback` and `mdmueller` was about debugging a failing Travis build.
  - The conversation was closed after discussing some documentation.

[R] ENH: ignoring comment lines and empty lines inCSV files

Distribution of comment subjects

Distribution of users commen

Swarmplot of comments over time

Sequence plot of comments ordered on time
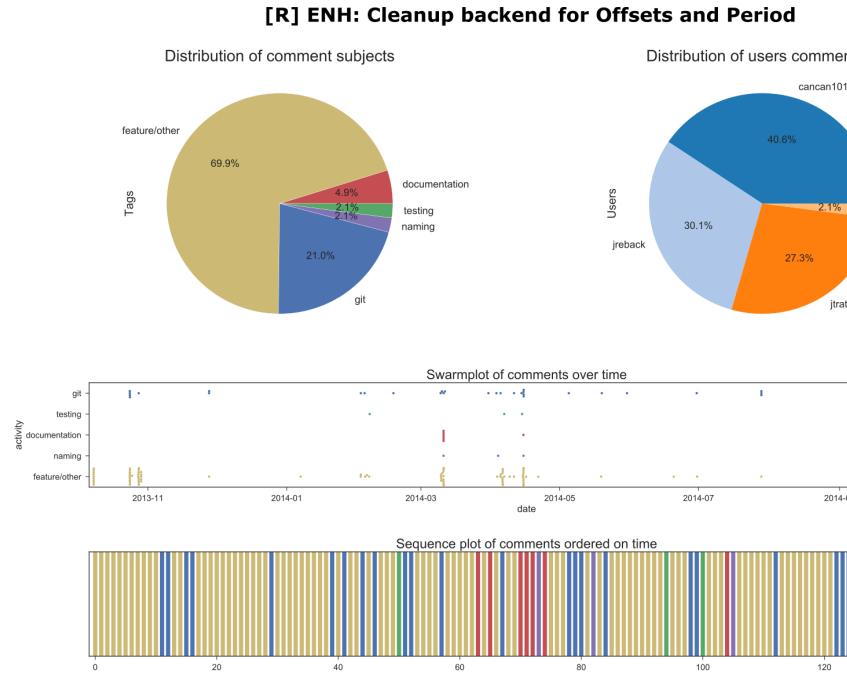
16.11.2.5.2 Summarizing Visualizations

A confirmation of the main discussion between `jreback` and `mdmueller`. The PR seems to be dotted with remarks on git related matters, which may be related to debugging the failing Travis build.

16.11.2.6 6. ENH: Cleanup backend for Offsets and Period #5148

16.11.2.6.1 Context

- Author: cancan101
- Timespan: 2013-10 : 2014-06
- Touches: Offsets and Period backend. Concerned a cleanup enhancement of the backend.
- Reason closed: Pull request fixed to many issues at once, requiring major rebase.
- Preceding issues: #5306, #5082, #5028, #4878, #5418.
- Reviewers: jratner (main), jreback
- Notes:
    - The user `nehalecky` briefly commented that they hoped to see the PR merged asap and thank cancan101 for their work.
    - `jreback` (only) appeared to close the PR, stating that a major rebase was required before acceptance and that they wanted separate PR's for each issue.
    - The main discussion was about finding a clean way to integrate the work made in cython and some refactorings to make the changes neater.

[R] ENH: Cleanup backend for Offsets and Period

Distribution of comment subjects

Distribution of users commen...

Swarmplot of comments over time

Sequence plot of comments ordered on time
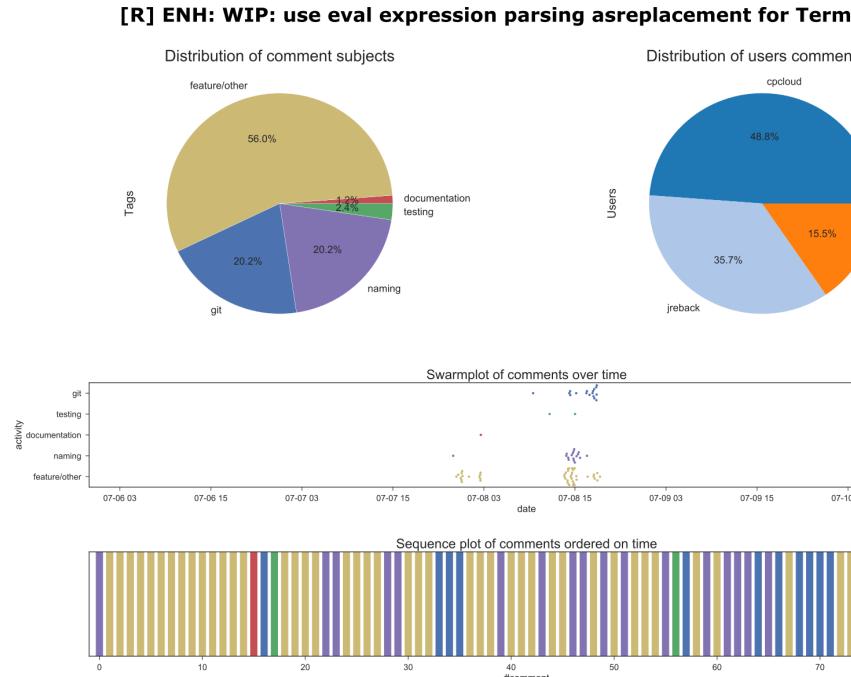
16.11.2.6.2 Summarizing Visualizations

The large amount of git discussion stands out, probably due to the major rebase that was required to merge this PR properly. Mainly `jreback`, `cancan101` and `jratner` discussing this. The PR was open for quite a bit of time, close to year, and activity seemed to happen mainly in bursts between March and May 2014, after which the PR lay dormant for quite a while longer.

16.11.2.7   7. WIP: use eval expression parsing as replacement for Term in HDFStore #4155

16.11.2.7.1   Context

- Author: jreback
- Timespan: 2013-07 : 2013-07
- Touches: HDFStore Term. Concerned a larger commit about extending the work of user cpcloud on eval expressions.
- Reason closed: Superseeded by another pull request: #4162.
- Preceding issues: PR#4037, #3393.
- Reviewers: cpcloud (main), meteore (comments)
- Notes:
    - The user `meteore` was a user of pandas interested in this change for their own use-cases, supporting the developers with additional insight and questions.

[R] ENH: WIP: use eval expression parsing asreplacement for Term

Distribution of comment subjects

Distribution of users commen

Swarmplot of comments over time

Sequence plot of comments ordered on time
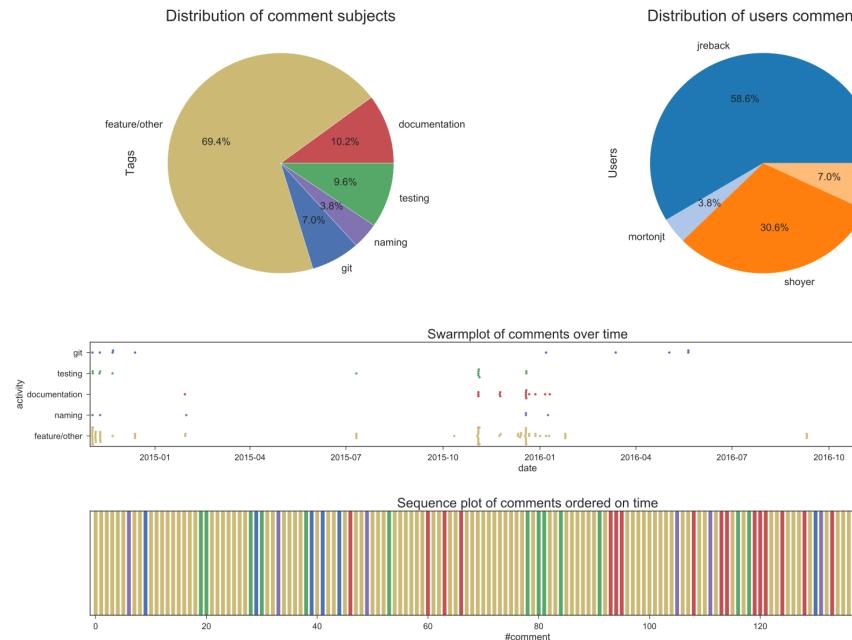
16.11.2.7.2 Summarizing Visualizations

A lot of comments on naming in this, but this is a false positive, as most of the naming-related comments were about the feature itself regarding the `name` field in one of the classes. The git discussion towards the end seemed to be about redirecting the efforts to a single PR on the subject to have everything in one place.

16.11.2.8   8. WIP/API/ENH: IntervalIndex #8707

16.11.2.8.1   Context

- Author: shoyer
- Timespan: 2014-11 : 2017-03
- Touches: IntervalIndex for expressing intervals. Concerned an enhancement for adding support for expressing intervals.
- Reason closed: Pull request could not be finished by shoyer due to lack of time.
- Preceding issues: #7640, #8625.
- Reviewers: jreback (main), jorisvandenbossche
- Notes:
    - BDFL wesm entered the discussion, suggesting to move the code to the new indexes package.
    - Closed on Sep 10, 2016 by `jreback` because `shoyer` indicated he did not have the time to finish the tests, documentation and Cython optimizations. `jreback` expressed interest in taking it over later on and continue working on it.
    - Most discussion was about the right way to express the interval, including technical performance concerns related to materialization of the dataframes.
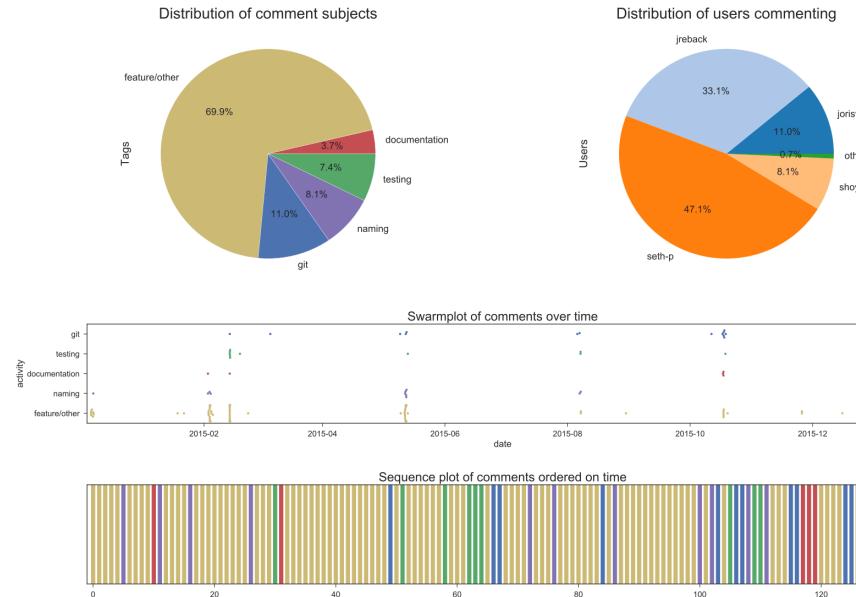
[R] WIP/API/ENH: IntervalIndex

16.11.2.8.2 Summarizing Visualizations

Main dicussion between `shoyer` and `jreback`, quite a bit of discussion on documentation, this seemed to stem from the fact that `shoyer` forgot some doc strings for his newly added features which was pointed out during code reviews.

16.11.2.9 9. ENH/API: DataFrame.stack() support for level=None, sequentially=True/False, and NaN level values. #9023

16.11.2.9.1 Context

- Author: seth-p
- Timespan: 2014-12 : 2016-01
- Touches: stack method of DataFrame
- Reason closed: Pull request could not be finished by seth-p due to lack of time.
- Preceding issues: #8851, #9399, #9406, #9533.
- Reviewers: jreback (main), shoyer, jorisvandenbossche.
- Notes:
    - Closed by `jreback` on Jan 20, 2016 because `seth-p` had no time to continue on it.
    - The latter half of the discussion was about setting up the virtual environent properly.
    - Most of the discussion was about the API implications. It was interesting to note that `shoyer` eventualy argued that an API change was preferred over a backwards-compatible 'hack'.

[R] ENH/API: DataFrame.stack() support forlevel=None, sequentially=True/False,

Distribution of comment subjects

Distribution of users commenting

Swarmplot of comments over time

Sequence plot of comments ordered on time
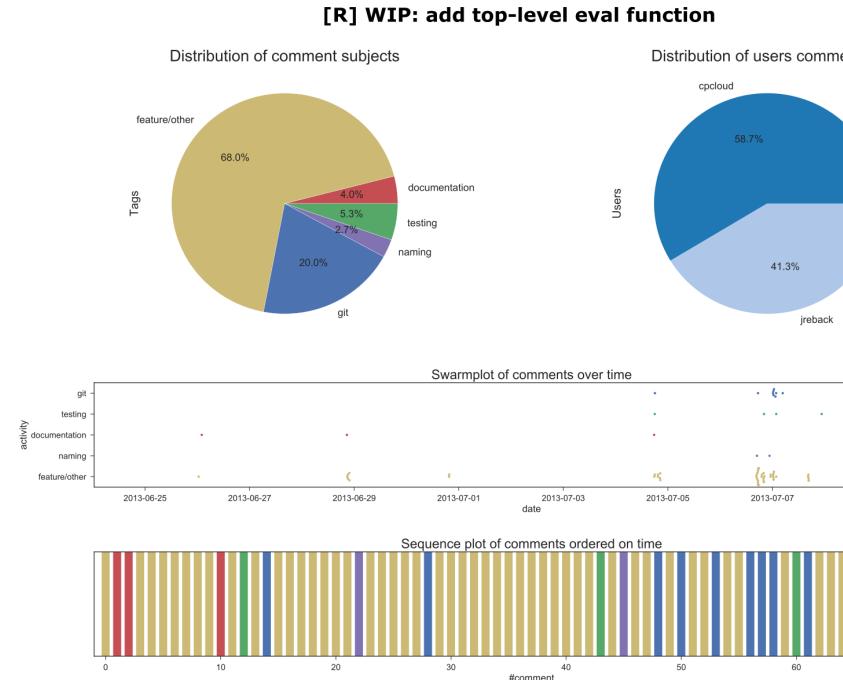
16.11.2.9.2   Summarizing Visualizations

Mainly `jreback` and `seth-p` discussing on this PR. A lot of git comments towards the end, mainly from rebase requests.

16.11.2.10   10. WIP: add top-level eval function #4037

16.11.2.10.1   Context

- Author: cpcloud
- Timespan: 2013-06 : 2013-07
- Touches: HDFStore Term
- Reason closed: Git history did not accurately depicted the history of contributions and hence another pull request was opened (probably this one).
- Preceding issues: #3393.
- Reviewers: jreback (main), wesm(minor), dragoljub (comment), jratner (comment)
- Notes:
    - Much discussion revolved around the finer workings of parsing/interpreting the operators in the eval function.
    - Quite some discussion also revolves around the usage of git, in particular `cpcloud` and `jreback` trying to access each other's work.
    - We can observe that some functionality that was previously in the *pandas.core.expressions* package was moved to *pandas.computation.expressions*.

[R] WIP: add top-level eval function

Distribution of comment subjects

Distribution of users comme

Swarmplot of comments over time

Sequence plot of comments ordered on time

16.11.2.10.2 Summarizing Visualizations
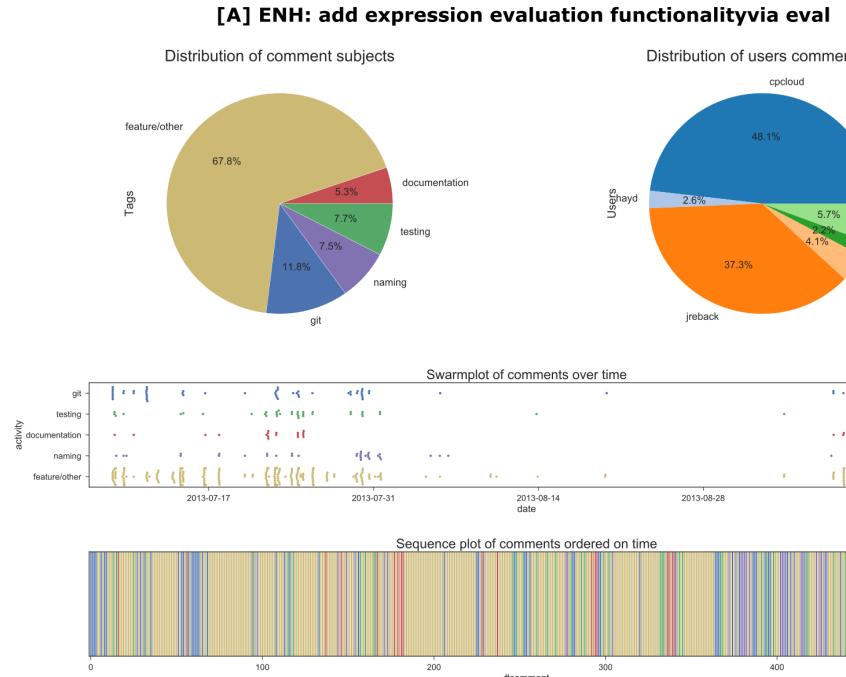
The only participants here were `cpcloud` and `jreback`. As with PR7, the git discussion towards the end seemed to be about redirecting the efforts to a single PR on the subject to have everything in one place.

16.11.3 Most Discussed Merged Pull Requests

16.11.3.1 11. ENH: add expression evaluation functionality via eval #4164

16.11.3.1.1 Context

- Author: cpcloud
- Timespan: 2013-07 : 2013-09
- Touches: not a lot, it mostly adds new functionality. It basically allows string expressions to be parsed with a custom backend/engine to efficiently evaluate operations on dataframes. Also a lot of documentation added as well.
- Preceding issues: #3393, #2560
- Reviewers: jreback, TomAugspurger
- Notes:
    - It seems that this PR was more of a roadmap/checklist to getting that feature done mostly, however a bit of functionality/refactoring was added during the course of the PR which was not covered by the checkboxes (e.g. QueryEngine).

[A] ENH: add expression evaluation functionalityvia eval
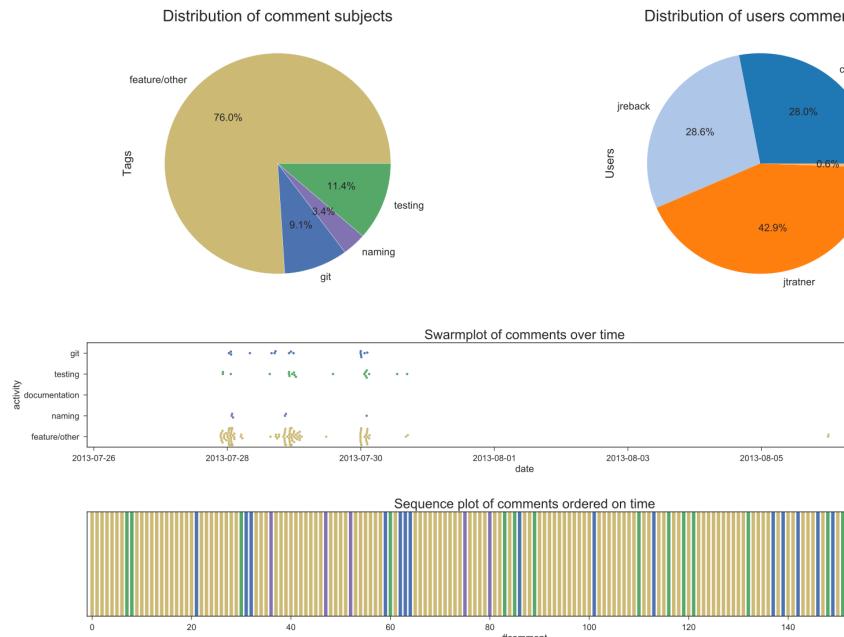
16.11.3.1.2 Summarizing Visualizations

Even though the PR was very large, the main discussion seemed to be between `cpcloud` and `jreback`, which makes sense given the previous PRs on this same feature were mostly discussed by them as well. The PR seemed to have a dead zone in the middle, with barely any activity, could be holiday-related.

16.11.3.2 12. CLN/ENH/BLD: Remove need for 2to3 for Python 3. #4384

16.11.3.2.1 Context
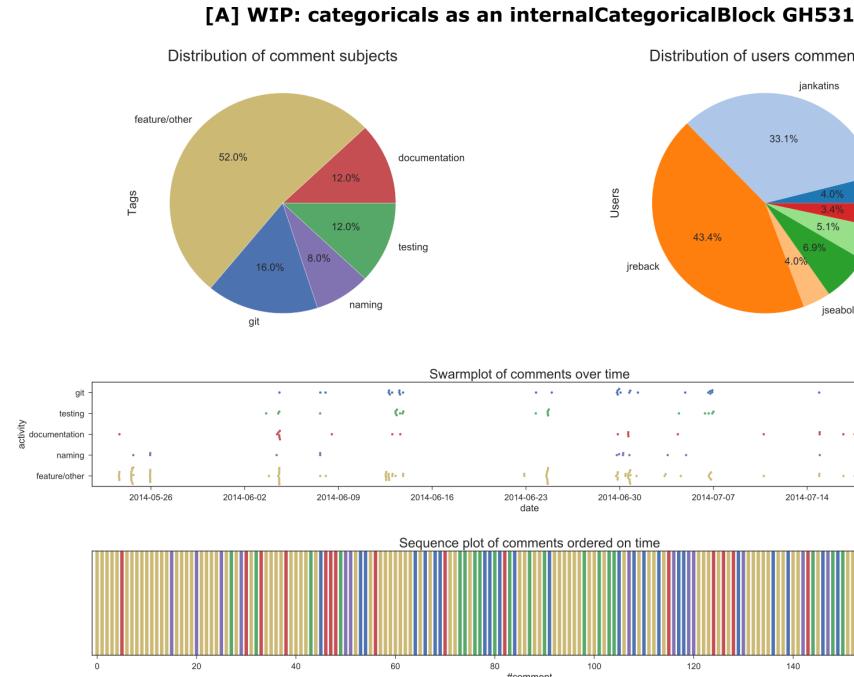
- Author: jtratner
- Timespan: 2013-07, 2013-08
- Touches: nearly the entire project, since it makes the code base compatible between python 2 and 3
- Preceding issues: #4375, #4372
- Reviewers: cpcloud, jreback
- Notes: n/a

[A] CLN/ENH/BLD: Remove need for 2to3 for Python 3

Distribution of comment subjects

Distribution of users commen

Swarmplot of comments over time

Sequence plot of comments ordered on time

16.11.3.2.2 Summarizing Visualizations

The main discussion here is between `jreback`, `cpcloud` and `jtratner`. Interestingly, most of the activity was in one month, before the holidays, and it got merged only after, hence the gap.

16.11.3.3 13. WIP: categoricals as an internal CategoricalBlock GH5313 #7217

16.11.3.3.1 Context

- Author: jreback
- Timespan: 2014-05, 2014-07
- Touches: any component in pandas that somehow manipulates or uses datatypes, such as the algorithms class, the dataframe class, etc. Also adds a new data class.
- Preceding issues: #5313, #5314, #3943
- Reviewers: jankatins, njsmith
- Notes:
    - It seems that there is quite a bit of discussion on naming, API etc, as such core team member `jorisvandenbossche` intervenes with a proposal for creating some sort of design document for pandas to get these kind of points clear.
    - `jankatins` calls for help from the 'statistics side': `jseabold`, `josef-pkt`, `cancan101`, `kshedden`, `upandacross` and `cfarmer`, which appear to have experience with R.
    - This feature is quite desirable among the userbase because it brings in features from its competitor R.

[A] WIP: categoricals as an internalCategoricalBlock GH531

Distribution of comment subjects

Distribution of users commen...

Swarmplot of comments over time

Sequence plot of comments ordered on time
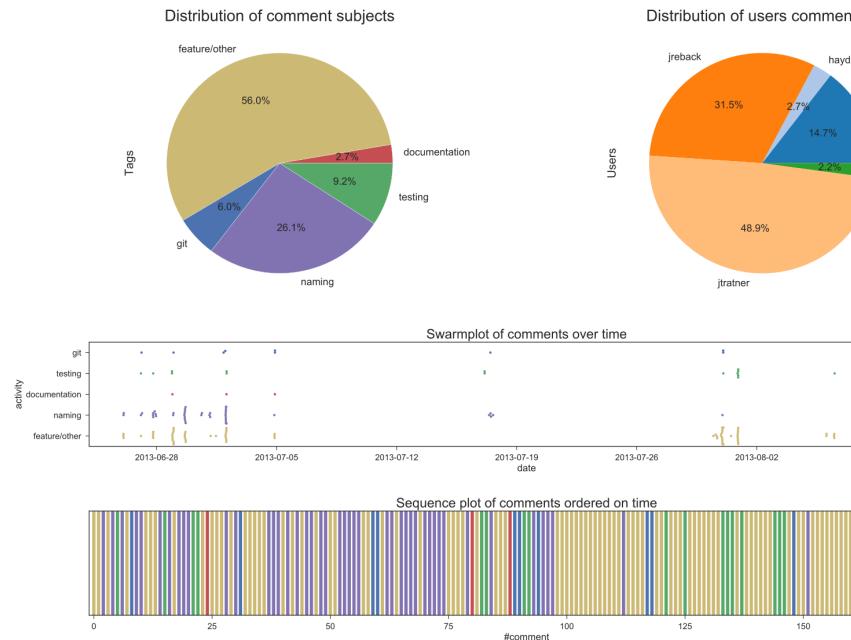
16.11.3.3.2   Summarizing Visualizations

This PR had quite a lot of people involved, with leading discussion by `jankatins` and `jreback`. Quite a lot of discussion was devoted to things like git, testing, documentation. The comments on documentation towards the end was apparently because they were broken.

16.11.3.4   14. ENH/BUG: Fix names, levels and labels handling in MultiIndex #4039

16.11.3.4.1   Context
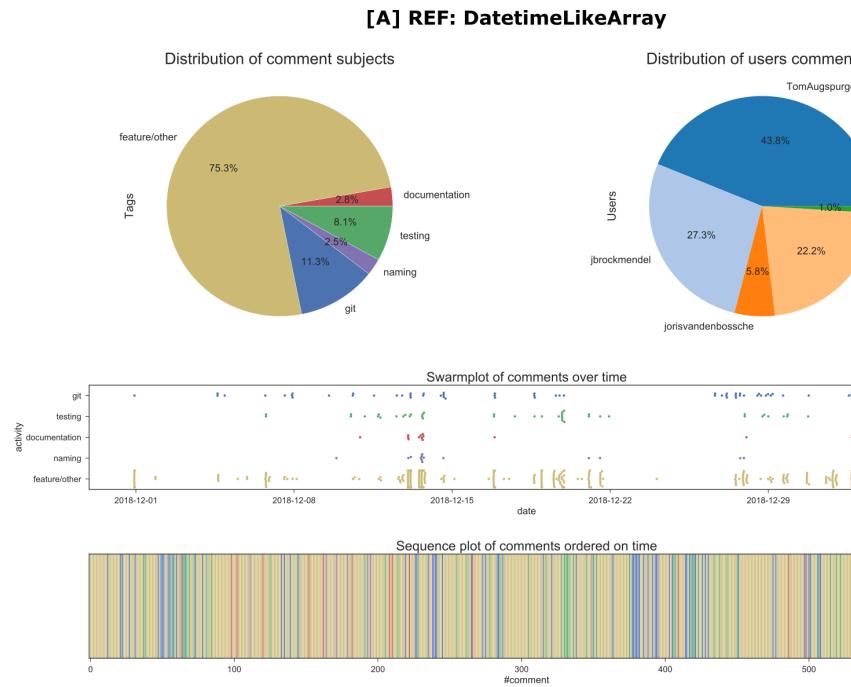
- Author: jtratner
- Timespan: 2013-06, 2013-08
- Touches: See PR description, it has a nice summary.
- Preceeding issues: #4202, #3714, #3742
- Reviewers: cpcloud, jreback
- Notes: n/a

[A] ENH/BUG: Fix names, levels and labels handlingin MultiIn

16.11.3.4.2 Summarizing Visualizations

Naturally, a lot of comments on naming, as the PR is focused on this. The main discussion is between `jreback` and `jtratner`, with `cpcloud` also chiming in. The first half of the PR is very focused on naming conventions, while the second half was more about levels/labels.

16.11.3.5 15. REF: DatetimeLikeArray #24024

16.11.3.5.1 Context

- Author: TomAugspurger
- Timespan: 2018-12, 2019-01
- Touches: anything related to dates, times and date/time differences.
- Preceding issues: #4202, #3714, #3742
- Reviewers: jreback, jbockmendel
- Notes:
    - `jorisvandenbossche` comes in with an issue and a proposal to start a new issue for discussing it, related to API-breaking changes.

[A] REF: DatetimeLikeArray

Distribution of comment subjects

Distribution of users commen

Swarmplot of comments over time

Sequence plot of comments ordered on time
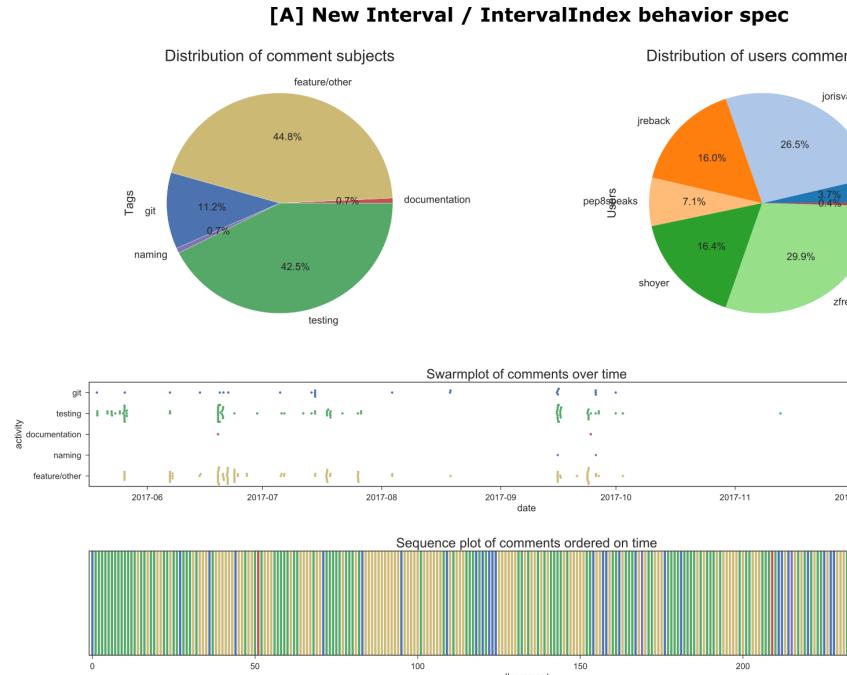
16.11.3.5.2   Summarizing Visualizations

Mainly `TomAugspurger`, `jbrockmendel` and `jreback` involved here. This PR had relatively many code
review comments, since this PR list is sorted by issue comments and it had 600 total comments.

16.11.3.6   16. New Interval / IntervalIndex behavior spec #16386

16.11.3.6.1   Context

- Author: zfrenchee
- Timespan: 2017-06, 2018-01
- Touches: mainly adds a lot of tests for the new more well-defined Interval/IntervalIndex specification.
- Preceding issues: #16316
- Reviewers: shoyer, jreback, jorisvandenbossche
- Notes:
    - There seems to be a lot of mentions of "xfails", which are tests that cannot succeed due to a bug
      not being fixed, or feature not being implemented.

[A] New Interval / IntervalIndex behavior spec

Distribution of comment subjects

Distribution of users commenting

Swarmplot of comments over time

Sequence plot of comments ordered on time
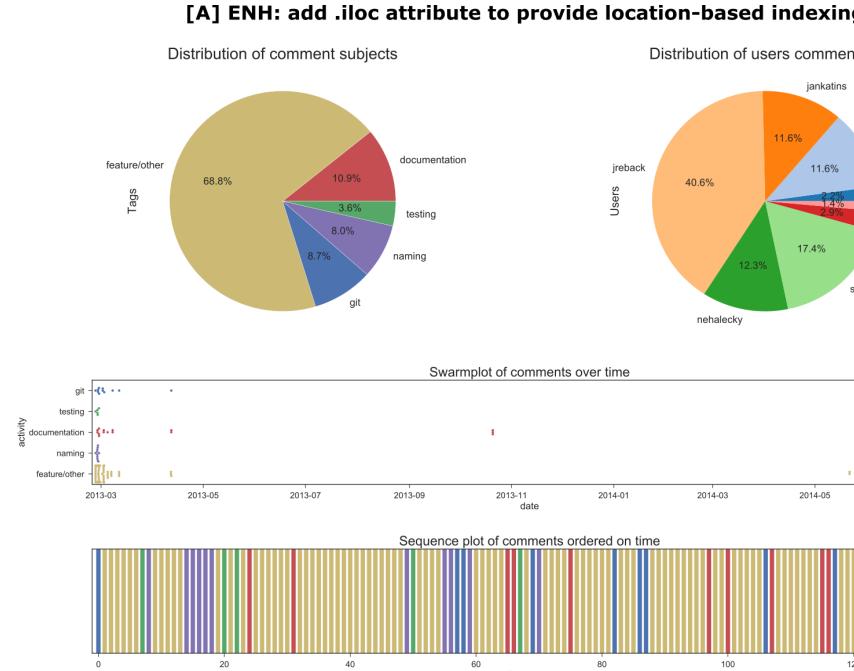
16.11.3.6.2 Summarizing Visualizations

A mixed discussion on the PR, between `zfrenchee`, `jreback`, `shoyer` and `jorisvandenbossche`. A lot of the comments were about tests, this was because there were more tests that needed to be added, some tests were at the wrong location, or some tests were not good enough. This PR also had some dips in activity around Nov/Dec before the team got back to it.

16.11.3.7 17. ENH: add .iloc attribute to provide location-based indexing #2922

16.11.3.7.1 Context

- Author: jreback
- Timespan: 2013-03, 2014-07
- Touches: dataframes, series, etc that will use these new location-based indexing features. Also adds test and documentation for them.
- Preceding issues: n/a
- Reviewer: stephenwlin, nehalecky, jankatins
- Notes:
    - This feature seems to be desirable among the users, as there were quite a few comments praising it.

[A] ENH: add .iloc attribute to provide location-based indexing

Distribution of comment subjects

Distribution of users commen
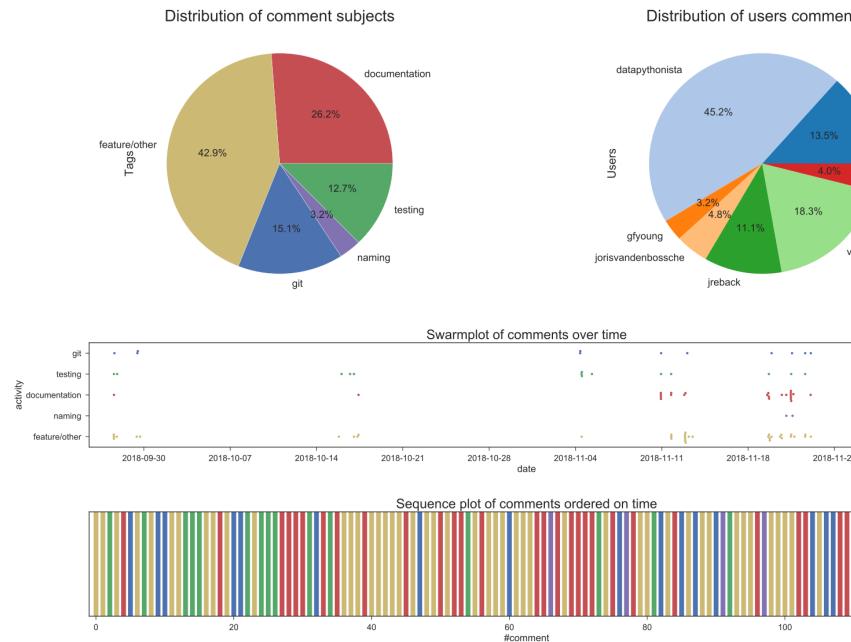
16.11.3.7.2   Summarizing Visualizations

Mainly `jreback` discussing this with `stephenwlin`, `jankatins` and `nehalecky`. Relatively little discussion on tests, but more than usualy on documentation. Also, most of this PR happened at the beginning, it only got merged half a year later.

16.11.3.8   18. CI: Linting with azure instead of travis #22854

16.11.3.8.1   Context
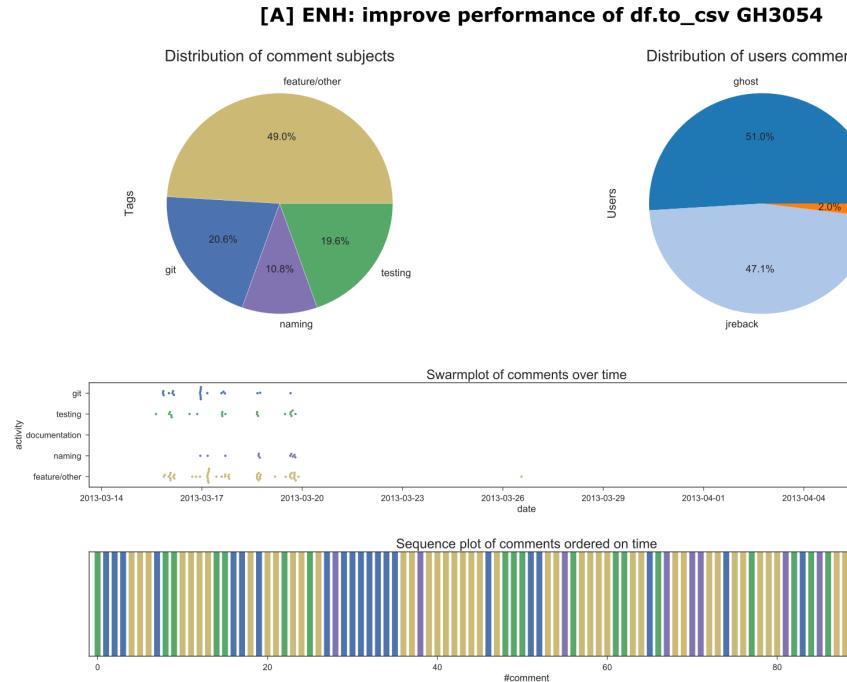
- Author: datapythonista
- Timespan: 2018-10, 2018-11
- Touches: the CI setup.
- Preceding issues: #22844
- Reviewer: TomAugspurger, vtbassmatt (CI assistance), jreback
- Notes:
    - It also has some minor changes in frame.py, panel.py, which seem not directly related, this may be an anomaly from a merge from master. Overall the git management seems inconsistent: sometimes a rebase is performed, while sometimes a merge is performed.

[A] CI: Linting with azure instead of travis

16.11.3.8.2 Summarizing Visualizations

Mainly discussion between `datapythonista`, `TomAugspurger` and `vtbassmatt`. The large ratio of documentation was a false positive, it came from references to the microsoft documentation for Azure. The PR had most of its activity a few months after creation.

16.11.3.9 19. ENH: improve performance of df.to_csv GH3054 #3059

16.11.3.9.1 Context

- Author: "ghost" (deleted user)
- Timespan: 2013-03, 2013-04
- Touches: the to_csv function of dataframes, massively improving its performance.
- Preceding issues: #3054
- Reviewer: jreback
- Notes:
  - This PR is by a deleted user.
  - The main discussion was by the "ghost" and `jreback`, both (playfully) competing with each other to find neat tricks to further improve performance.

[A] ENH: improve performance of df.to_csv GH3054
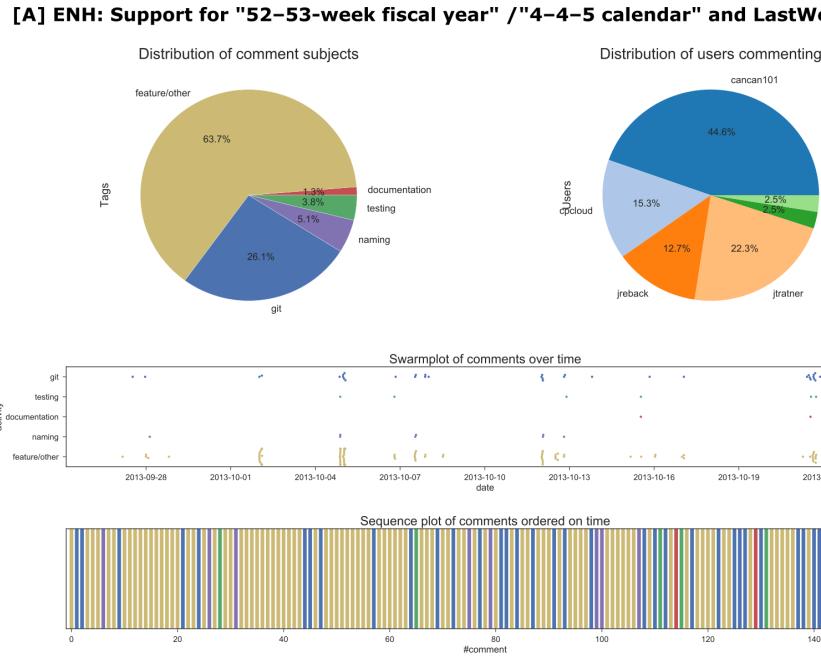
16.11.3.9.2   Summarizing Visualizations

Almost exclusively `ghost` and `jreback` discussing this. The comment towards the end was from an enthusiastic user.

16.11.3.10   20. ENH: Support for "52–53-week fiscal year" / "4–4–5 calendar" and LastWeekOf-Month DateOffset #5004

16.11.3.10.1   Context

- Author: cancan101
- Timespan: 2013-09, 2013-10
- Touches: timeseries.py and frequencies.py, adding some functionality and fixes, and also adds tests for the new functionality for irregular calendars.
- Preceding issues: #4511, #4637
- Reviewers: jreback, cpcloud, jtratner
- Notes: n/a

[A] ENH: Support for "52–53-week fiscal year" /"4–4–5 calendar" and LastWe

16.11.3.10.2 Summarizing Visualizations

Main discussion between cancan101, cpcloud, jratner and jreback. The heavy git discussion towards the end was again because of rebase requests.

16.11.4 Visual Summary Retrospective

A global visualization might potentially reveal more interesting information.

16.11.4.1 Global Visual Summary

Here we can see that indeed jreback has the largest share of comments by far, which is confirmed by his important integrator role over the course of the project. He has commented in all of the large PRs that we have investigated. Other than that, jratner, TomAugspurger and cpcloud also show significant involvement in the large PRs. Furthermore, the swarm plot shows that there is a gap in activity between 2018 and 2019 when it comes down to big pull requests, while 2013, 2014 (coinciding with 0.11, 0.12 and 0.13) and end-2018 (mainly from the DateTime-like array PR, number 16) show the largest amount of activity. The sequence plot does not seem to be very useful due to the sheer amount of comments (and the discontinuity of the comment counter), but does reveal the shape of the subject distribution.

Overall, a quarter of all the comments are on git or testing, with a relatively smaller share on documentation or naming. Git usually gets mentioned because of rebase requests or because of (resulting) general issues with managing git for the particular PR. Testing is often an important subject in PRs that add new functionality, it's often either a failing test or a missing test that is being discussed.
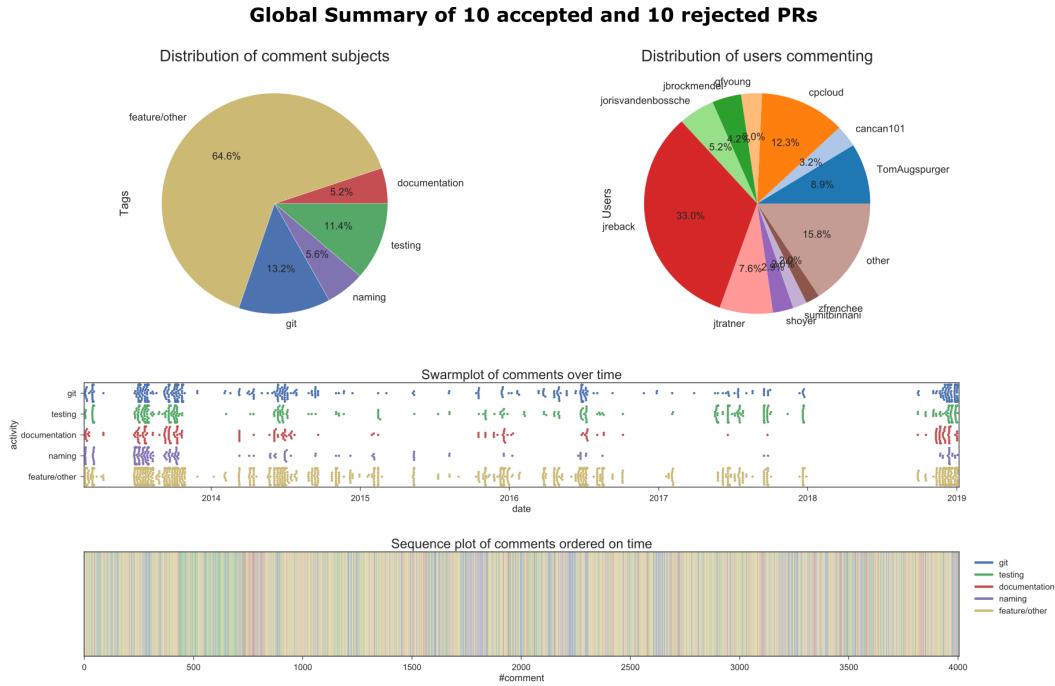
Figure 16.1: Summarizing Visualizations

16.11.5   Other Information/Communication Sources

16.11.5.1   Gitter

On Gitter, some new names can be found, but also some familiar ones. The latest discussion seems to be usage related rather than dev. There was however a question before by `TrigonaMinima` about tests, which was answered by `jorisvandenbossche`. Before that `jreback` also participated in some discussion with `jorisvandenbossche`, about some change by `jorisvandenbossche` that `jreback` disagrees with. Another contributor by the name `scari` mentions an issue with timestamps.

More interestingly, upon a question about the dual travis/azure devops setup by `fuglede`, `jorisvandenbossche` answered that this was mainly `datapythonista` that made this switch to **Azure**, apparently with help from the Azure team itself. The reason was apparently that Azure was faster and provides more (parallel) builds compared to Appveyor.

16.11.5.2   Mailing Lists

Not too much new information here, aside from **Vaex** as a library that uses Pandas. Release posts are also often put on the mailing list, with change logs for the new version. Interestingly, Wes Mckinney is still quite active on the mailing lists post-2014, after his primary contributions. Mainly posts about re-organizing code (e.g. a way to split up mega-module such as frame.py). Later discussions are a lot about future refactorings

(pandas 2.0), also initiated by Wes. There's also mention of starting a new communication platform/hub for all pydata open source projects (e.g. pandas, matplotlib, etc) via a service like Discourse, just to discuss

16.11.5.3 Conference Articles/Videos/Presentations

16.11.5.3.1 Pandas Summit 2019 (Feb 2019) This article is a retrospective on Pandas Summit 2019 by dunnhumby. In the article it is explained that dunnhumby contributed financially to a conference where the proceeds went to **NumFocus**, which is an organization that supports Pandas. It also appears that PRs must be approved by two core team members, and that they may receive up to 10 PRs a day. Apparently there is one contributor that nobody has met in person but works a lot on Pandas.

There is also a picture of a slide from Marc Garcia (`datapythonista`) that contains three dev members that are labeled as core team members: `Joris van den Bossche`, `Pietro Batison`, `Marc Garcia`.

16.11.5.3.2 PyData London Meetup #47 (Aug 2018) Main finding here is that the founder Wes Mckinney seems out of touch with pandas according to Marc Garcia (`datapythonista`), because he went on to create **Apache Arrow**, which is a unified columnar processing/storage system which is supposed to remove the overhead of converting data formats between the data analysis libraries. This means that Apache Arrow can actually be used by Pandas (and Pandas already has support for it regarding data transfer). We can also find some other possible "competitors", such as the **Julia** data science language and the **Apache Spark** library. Future plans: pandas seems to want to support Python 3.7 properly, and especially focus on a lot of maintenance. In fact, they want to drop Python 2 support in 2019, together with matplotlib and other libraries that are also dropping support for it.

16.12 Appendix D: Code Coverage

Table 3: Code coverage reported by Codecov as of commit 707c7201a744c48c835f719421e47480beee2184

Files	Tracked Lines	Covered Lines	Missed Lines	Coverage
_libs	7	7	0	100.00%
api	8	8	0	100.00%
compat	649	411	238	63.32%
core	36.189	33.901	2.288	93.67%
errors	12	12	0	100%
io	10.078	9.139	939	90.68%
plotting	2.830	2.058	772	72.72%
tseries	1.657	1.598	59	96.43%
util	1.547	1.208	339	78.08%
Totals (172 files)	**52.977**	**48.342**	**4.635**	**91.2%**

16.13   References

By Damy Ha, Nando Kartoredjo, Teodor Nikolov, and Xingchen Liu. Delft University of Technology.

Chapter 17

Poco

17.1   Table of Contents

17.2 Introduction

The Portable Components C++ Libraries (POCO) project was started by Günter Obiltschnig in 2004. During that time C# and .Net, were very popular. As a compiled language, C++ reached its lowest point at that time. Günter Obiltschnig believed in C++, and was committed to create a high-quality, easy to use C++ library which is focused on providing networking libraries for the creation of different networking configurations.

The first release of POCO was in 2005, and since its release, more than 180 developers have contributed to the source code of it. POCO is an open source, cross platform software project written in ANSI/ISO standard C++ and based on the C++ Standard Library. Due to its portability, modularity, and efficiency, POCO is widely used for network-centric and portable applications development. Applications written with POCO can be easily ported to a new platform, as a result of the platform independencies of POCO. As a non-profit open source project, POCO is completely free for both the commercial and non-commercial use.

This report investigates the software architecture of POCO and contains six sections. We start with the Stakeholder Analysis, which lists the stakeholders of POCO and their respective stake. The next section is the Context View, which looks into the relationships, dependencies, and interactions between POCO and its environment. After the Context View, the Pull Request Analysis is presented, which summarises our findings of 20 analysed pull requests. The following section is the Development View, which describes the architecture that supports the software development process of POCO. The next section is the Functional View, which describes the architectural elements of one of the functions provided by POCO. The last section is the Technical Debt, which discusses the concept of Technical debt and its implications on the project. A summary is provided in the conclusion.

Version 1.9.0 was used at the time this report was written.

17.2.1 Stakeholders Analysis

The POCO project recognises a variety of stakeholders, each having their own share in the project. First, we would like to identify each stakeholder by its class, and second, we would like to illustrate the stakeholder involvement by making use of a *power/interest grid*.

17.2.1.1 Stakeholder Classes

The following classes can be identified within the POCO project.

17.2.1.1.1 Acquirers The acquirers are identified as the leading developers Günter Obiltschnig and Aleksandar Fabijanic. Both acquirers also act as the communicators of the project, interacting with pull requests and email traffic personally.

17.2.1.1.2 Assessors The POCO project has been governed by a *Technical Steering Committee* (TSC), which consists of the acquirers and a group of individuals selected by the committee which made significant and valuable contributions to the POCO project. All pull requests must be reviewed and accepted by a collaborator or a committee member. The collaborator is required to have sufficient expertise and is required to take full responsibility for the change made in the approved pull request.

17.2.1.1.3 Competitors The POCO project is widely used, mostly because of the easy of use of networking libraries for embedded devices, service applications, desktops, and many other applications. Therefore POCO also has its competitors. For example, ADAPTIVE Communication Environment (ACE) provides network programming libraries used for message routing, dynamic (re)configuration of distributed services, concurrent execution and synchronization. Another competitor is Boost which provides portable C++ source libraries, including libraries for networking like Boost.Asio, or Boost.Beast.

17.2.1.1.4 Developers The developer community consists of more that 180 developers which have contributed to the POCO project. Based on the information gathered from GitHub the people with the highest contribution are described in Appendix A.

17.2.1.1.5 Maintainers Applied Informatics Software Engineering GmbH, Applied Informatics for short, is the original developer and current maintainer of the POCO C++ Libraries. The acquirer Günter Obiltschnig is the founder of this company. In addition, managing the releases and maintaining the compatibility with different platforms is done by Francis Andre.

17.2.1.2 Testers

The POCO project doesn't have dedicated testers, instead it relies on the community to frequently test and maintain the project. This results into overlap between testers and developers, thus both are not completely independent. The testers with the highest contribution are described in Appendix B.

17.2.1.3   Users

The POCO project is a widely popular project within the open source community. At the moment of writing, at least two million *code results* can be found on GitHub for mentioning the POCO project within an open source repository. It is also included in the list of open source C++ libraries as one of the useful libraries by cppreference.com. A list of commercial users is also stated by POCO themselves, which include companies like Cisco, Oracle, and Panasonic.

17.2.1.4   Sponsors

The POCO project relies on sponsors to pay for the upkeep of the project's website and other platforms. Alternatively, companies are welcome to fund the development of a specific feature. To become a sponsor, companies are required to sign an agreement with Applied Informatics. Sponsors receive acknowledgement by either a display of their logo on the POCO project website, or by a brief mention of their contribution to the project. The POCO project is at the moment of writing actively supported by Schneider Electric and Siemens, which both are also users of the POCO C++ libraries.

17.2.1.5   Contact Persons

The POCO project provides several forms of communication. For purchased commercial support, the users are able to submit a message via the POCO website and get direct contact with the Applied Informatics support team. Sending an email to the support team is also an option. For non commercial support and other technical questions, POCO has a community scattered over multiple portals including Stack Overflow and Github, from which users can receive help from the wide POCO community. The latest updates and notifications related to the project can be also found on social media. Social media accounts are updated by Günter Obiltschnig. Technical questions or bug related issues can be addressed in the GitHub repository where a member of TSC could take action.

17.2.1.6   Service Partners

The POCO project provides services to other companies through their C++ libraries, which are essential for the products these companies deliver. A few examples are: HORIBA which uses POCO in automotive test systems, Nucor Steel which uses POCO in beam mill automation application, and Starticket which uses POCO in a ticketing and entrance control system running on an embedded Linux platform.

17.2.1.7   Integrators

The main responsibility of the integrators is to decide whether a commit to the project should be accepted or rejected based on the code quality, and to communicate the modifications to the source code with the contributors. The contributors of POCO must follow the Applied Informatics coding style standard and project requirements. The main integrators are Günter Obiltschnig and Aleksandar Fabijanic. Either one or both of them are involved in every pull request. Furthermore, the integrators are supported by a member of TSC. In case tests from the commited change are not passed or do not meet the requirements for coding style, the contributors are contacted and improvements are suggested.

17.2.2 Stakeholder Involvement

The figure below shows a *power/interest grid* which illustrates the the power a stakeholder has versus the interest it has in the success of the project. Each part of this grid can be divided into the following prioritisation categories:

- High power, high interest: **manage closely**: great effort should be made to satisfy requirements of the stakeholder, and a close relation should be maintained and managed.
- High power, low interest: **keep satisfied**: some effort should be made to satisfy the stakeholder. No more attention then needed is required, as they might lose even more interest.
- Low power, high interest: **keep informed**: keep the stakeholder informed as they might provide useful contribution to the project.
- Low power, low interest: **monitor** keep track of their activities but don't bother them with additional attention.

For each of the stakeholder classes, we will visualise their involvement and possible prioritisation using the following grid:

17.3 Context View

The context view describes the relationships, dependencies and interactions between the POCO project and it's environment. The context view starts off by stating the boundaries of POCO and then present the context model.

17.3.1 System scope

POCO has the following main responsibilities:

- POCO provides C++ libraries for command line/server applications, network programming, XML and JSON processing, database access and many more features.
- POCO can be used on many platforms including desktops, embedded and cloud applications, etc.
- POCO implements core features such as: classes for dynamic typing, cache framework, etc.
- POCO implements additional features including: compression, cryptography, database access, a file system, logging, multithreading, networking, processes and IPC, streams, text encoding, XML and JSON parsing.
- POCO provides unified access to different databases but doesn't implement the interface of the databases. Instead, it relies on third party connectors.
- POCO provides support for network-based applications but doesn't implement the Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols. Instead, it relies on OpenSSL.

17.3.2 Context Model

The figure below shows the context model of *POCO*.

The following entities can be found:
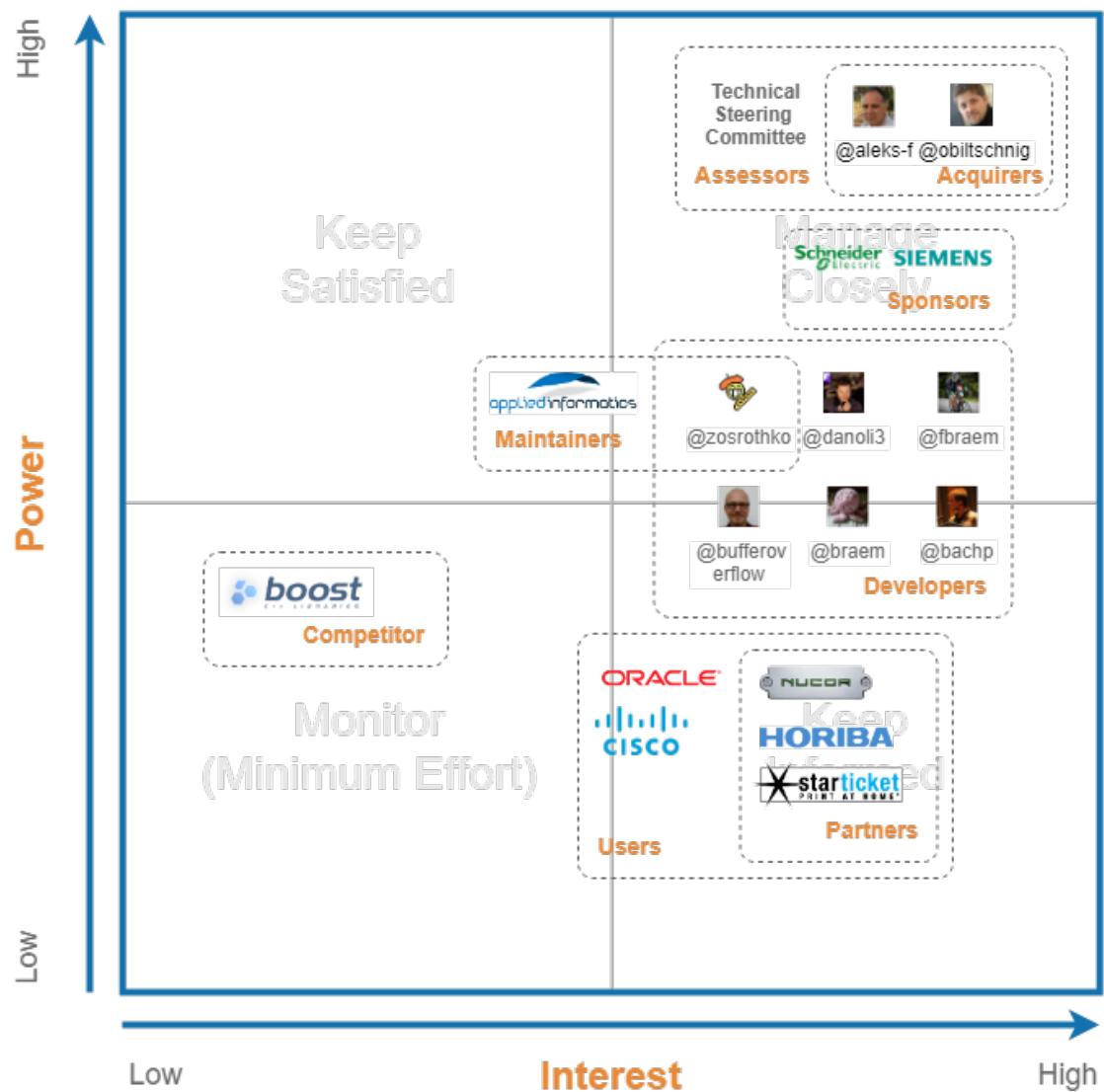
Figure 17.1: *power/interest grid* of *POCO*
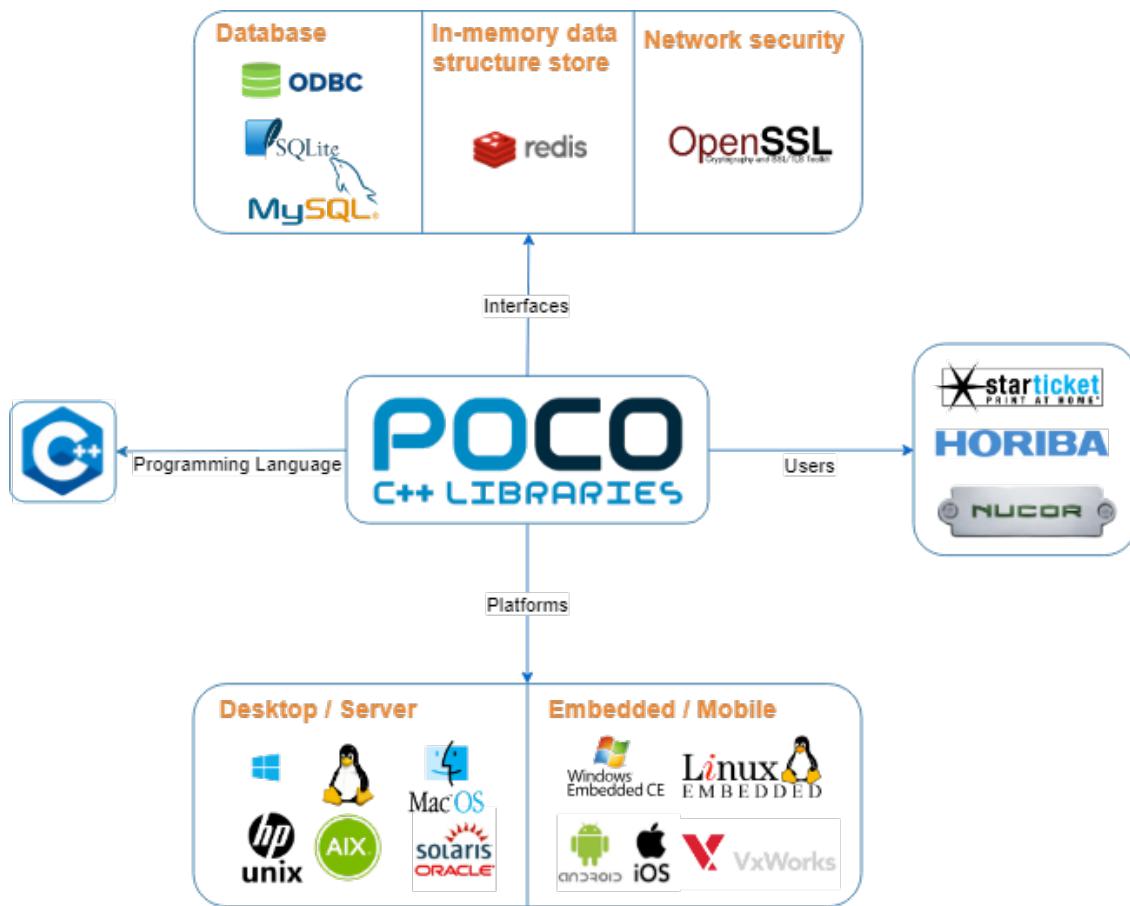
Figure 17.2: Context model of *POCO*

- **External Interfaces**: *ODBC*, *SQLite*, and *MySQL Client* provide an interface to access databases. *Redis* can be used as a database, cache, or message broker. *OpenSSL* is a toolkit for the TLS. Dependencies occur when specific libraries are used.
- **Users** : A system implemented by the users, or users them selves, are expected to provide and receive data and services to and from POCO.
- **Platform**: POCO covers a wide range of platforms. Renowned platforms are listed in the image.
- **Programming Language**: The programming language used to write code is C++. The libraries do support the processing of other languages such as JSON and HTML.

17.4   Pull Request Analysis

This section describes the general procedure of processing merge requests.

17.4.1   Accepted Requests

Most merge request are inspected by a (single) member of the TCP. The requestee and the TCP member discuss, on a high level, what the implemented feature should do and how it has been implemented. The TCP member gives a first impression of the code. Sometimes a solution has a fundamental problem, e.g. due to the proposed structure of the library, or due to development in other branches. If they have reached consensus, the real review takes place. The TCP member and a member of the development team take a look at the code and review it. Once consensus has been reached between the three parties, the code is merged.

Two events that regularly are that the TCP members almost always react on the same day and sometimes when Travis CI or AppVeyor (continuous integration services) gives an error, the branches are still merged. Multiple reasons could cause a continuous integration failure like a segmentation fault.

17.4.2   Rejected Requests

In general, there are four reasons why a request might be rejected. The first one is the rejection of the change proposed in the pull request. Both the requestee, and a member of the TCP will discuss the resulting impact, both negatively and positively. However, it is a member of the TCP which makes the final decision. The second one is the rejection of requests which add features to a frozen development. Specifically, some features had been rejected due to the oncoming release of version 2.0.0 of the POCO project. The third one is the rejection of pull requests which are in such a way problematic, that the current pull request is closed, and later in time a new one with a similar proposed change is created. The latest one is the rejection of pull requests where the change had already been fixed by another pull request. Commonly a discussion starts due to problems caused by the proposed change. After some discussion, a participant will point out that these problems had already been fixed in another pull request.

17.5   Development View

This section describes the architecture that supports the software development process of POCO. It's split into three parts. First the module organisation is described. Second, the common design model which states

the design constraints in order to maximise commonality is described. And finally, the codeline model which defines the organisation of the source code, the building, testing, and release process of the system is described.

17.5.1 Module Organisation

The four core libraries of POCO are: Foundation, XML, Util, and Net. Additional libraries are also available. The libraries are made up of smaller packages and modules. The important libraries and the dependencies are shown below.
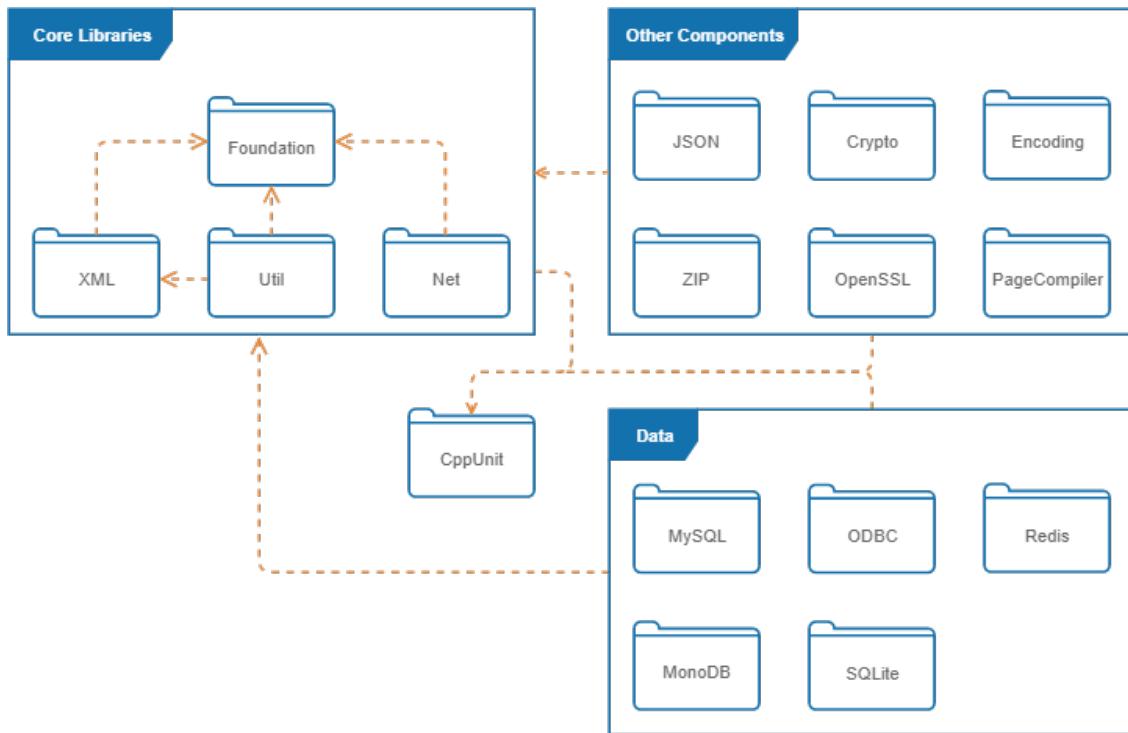
Figure 17.3: Library organisation of *POCO*

The following can be deducted from the libraries:

- The Foundation library contains the underlying platform abstraction layer and frequently used utility classes and functions. All other modules (except CppUnit) depend on the foundation library.
- The CppUnit module is a port from JUnit, a testing framework for Java. The module allows unit testing. All modules contain tests and thus depend on this module.
- The XML library handles the reading, processing and writing of XML.
- The Util library contains a framework to create command line/server applications. It supports command line handling and managing configuration information.
- The Net Library is used to write network based applications.

- The data libraries provide the interfaces to databases. The data libraries depend on the foundation library to provide the support of the underlaying platform.
- All other libraries implement addtional functions such as SSL support, JSON parsing and cryptography.

There are over 60 modules inside POCO. For this reason we've decided further investigate the module structure to the most important library (Foundation). The figures below shows the dependencies of the modules in the Foundation library.

A module is represented as a node and the number shows the dependencies within the foundation library. Internally the `Logging`, `Core` and `Filesystem` modules are used more frequently. The modules can't be layered, because the modules are too intertwined.

The bubble graph shows the modules outside of the foundation library that depend on this module. Externally the `Core` module is by far the most depended on.

17.5.2   Common Design Model

The common design model defines the design constraints in order to maximise commonality. This section is divided into three parts namely: the common processing elements the design, the standardisation of design elements, and the common software in the system.

17.5.2.1   Common Processing

Common processing entails the software elements in a design that benefit from standardisation. The most important standardisation in POCO are:

- POCO expects the users to define their own logging structure. The `Logger` class helps the user to achieve this. The `Logger` class supports 8 different types of priorities including: `PRIO_TRACE`, `PRIO_DEBUG` and `PRIO_INFORMATION`. A threshold value can be set such that only high level priorities are passed on.
- The Foundation core provide most of the functionalities that the libraries require. Using the Foundation library keeps data structures consistent over the libraries.
- Libraries have been written to access databases/memory structures (e.g. MySQL/Redis/MongoDB). These libraries should be used to connect to databases.
- A networking library is used to access the functionalities of OpenSSL.
- A cyrptography library is available to provide secure network engineering.

17.5.2.2   Standardisation of Design

The source code should adhere to the the CppCodingStyle guide. The libraries should have as few dependencies as possible between each other with the exception that the foundation library may always be used. The foundation library provides the underlying platform abstraction layer after all.

There is no documentation that states the design patterns that are used in the POCO library, but common design patterns such as singletons and factories have been found in the code.
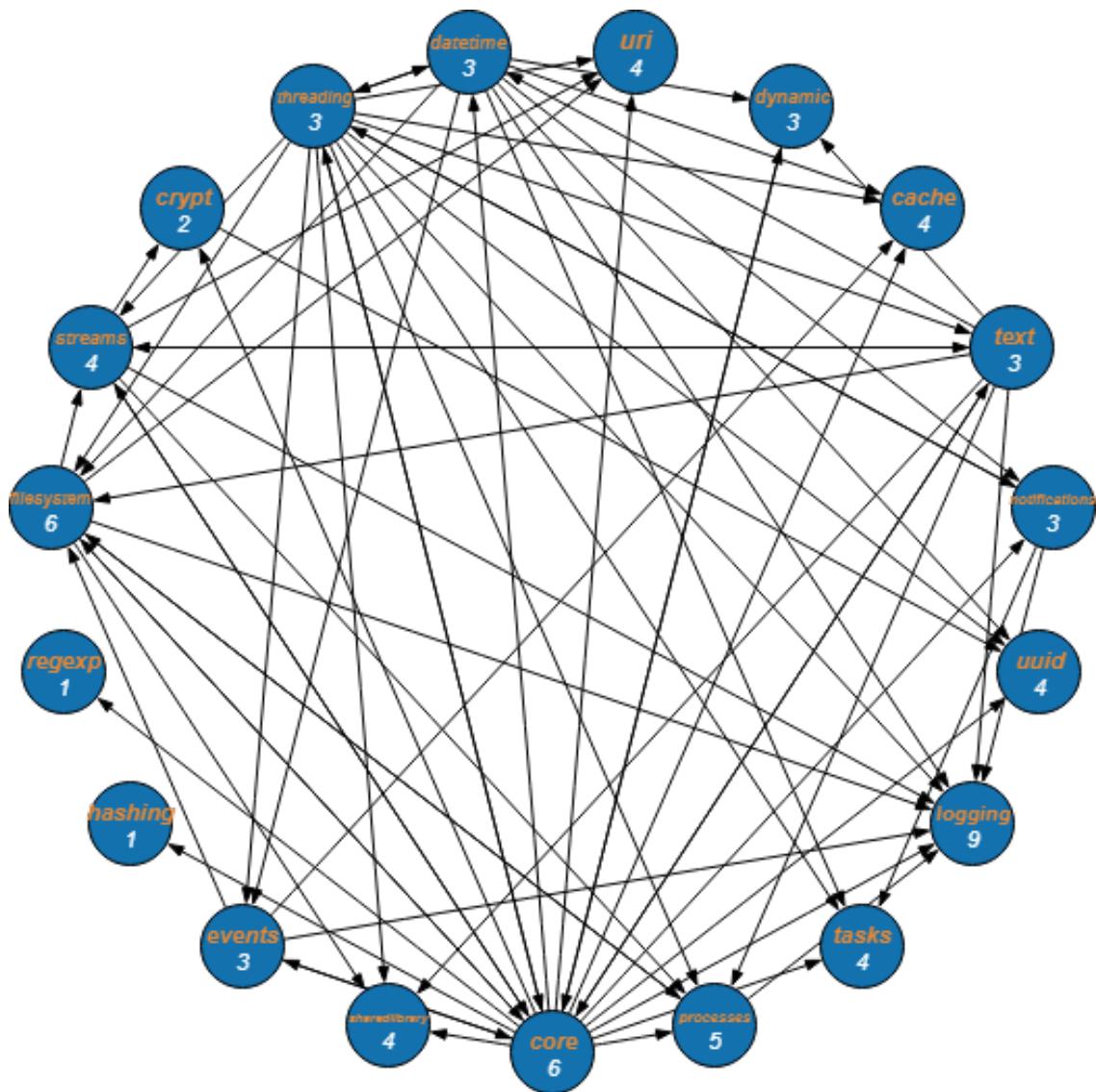
Figure 17.4: Module organisation of Foundation
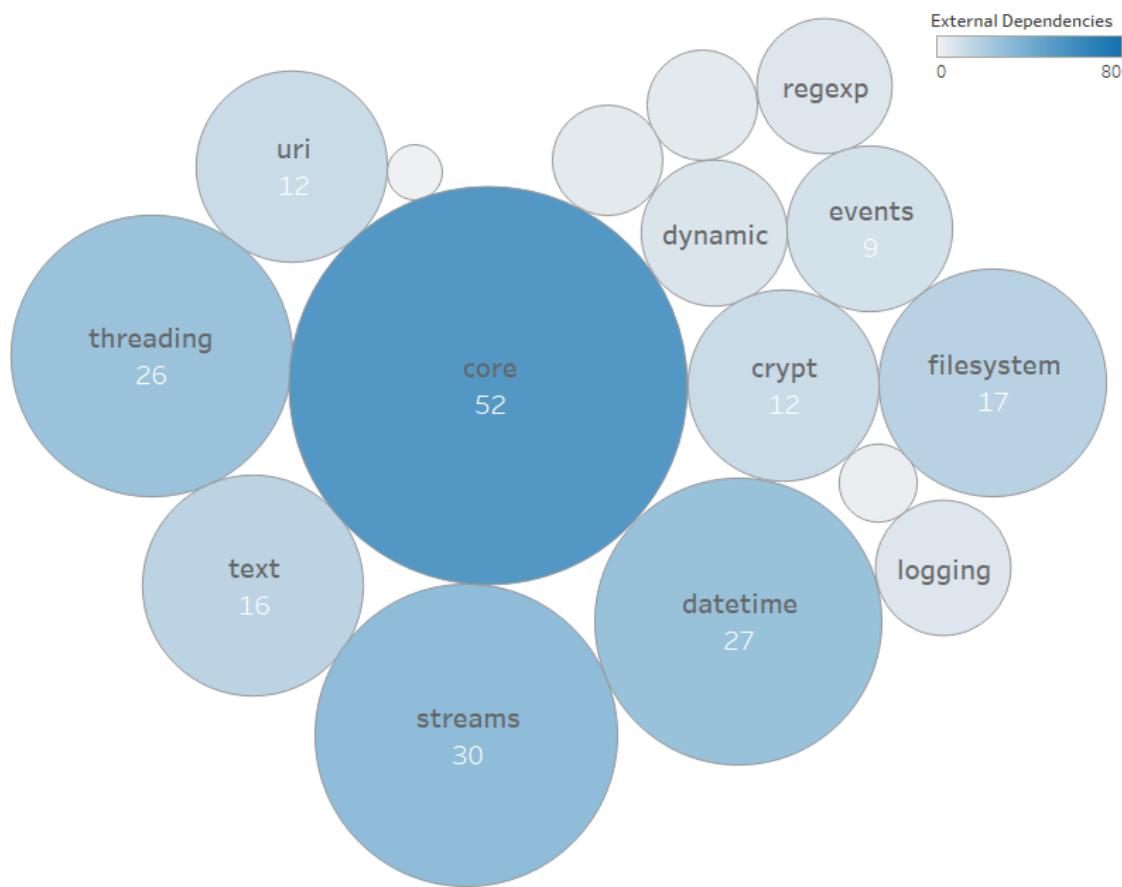
Figure 17.5: Module organisation of Foundation

17.5.2.3 Common software

The important libraries that contain common software in POCO are: the `foundation` library on which all other libraries are built on (e.g. message logging and instrumentation), the `Net` library which is use to create network based applications, and the `CppUnit` library which provides the framework for unit testing and the `database` libraries which provide access to databases.

17.5.3 Codeline model

The codeline model defines the organisation of the source code and the protocol to build, test and release the system. It includes the codeline organisation, the test procedure of candidate releases and the build, test, and release procedure.

17.5.3.1 Codeline Organisation

The develop branch of the POCO library is stored according to a structure. The branch is divided into 33 folders of which 12 folders are essential. The figure below illustrates the source code structure of POCO.
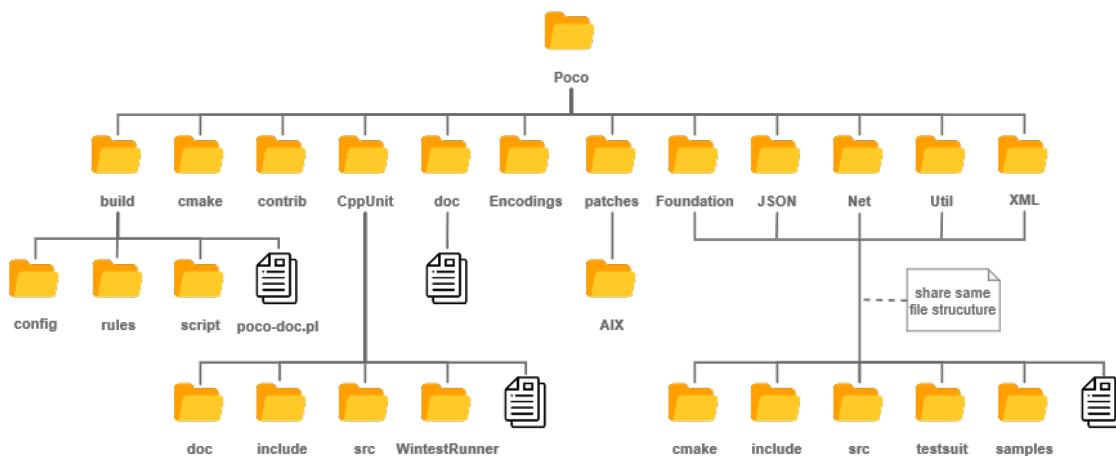
Figure 17.6: Codeline organisation of *POCO*

Some notable things of the file structure are:

- `build`: This folder contains three folders, `config`, `rules` and `script`. The files in this folder are used to build POCO on different platforms.
- `cmake`: Most of the files here are `.cmake` files. Besides the `cmake` file, two `cmake.in` files and one `.cpp` file are present.
- `contrib`: This folder contains a script to make the code suitable for Doxygen.
- `CppUnit`: This folder contains project files for various platforms to support the unit test framework.
- `doc`: This folder contains basic documentation and the tutorial of POCO.

- `Foundation`, `JSON`, `Net`, `Util`, and `XML`: These library folders contain make/build files, the header and source files for both the implementation and test suite in separate folders. Each library is also required to supply at least one example code.

17.5.3.2   Standardisation of Testing

After implementation of a feature, the code must be tested by the user. The contribution file states that contributors are required to write tests. The tests must pass, however the coverage of the tests is not defined.

The contributor is then supposed to test the implementation on a major platform (Linux, Windows, and Mac). The document does not define what testing means. The POCO library contains example codes that demonstrate the library and it's function. We assume that at minimum, testing implies the successful execution of these example codes. If there's no relevant example code available, an example code must be written.

In addition to manual testing, POCO also makes use of continuous integration services (Travis CI and AppVeyor). In general all checks must be passed.

17.5.3.3   Release Process

The POCO project, started by Günter Obiltschnig, was first released in February 2005. Aleksandar Fabijanic later joined the team as a contributor. The releases of Poco is numbered by the semantic versioning. Since its first release, more than 50 versions have been released. The timeline of the main release version is shown below.
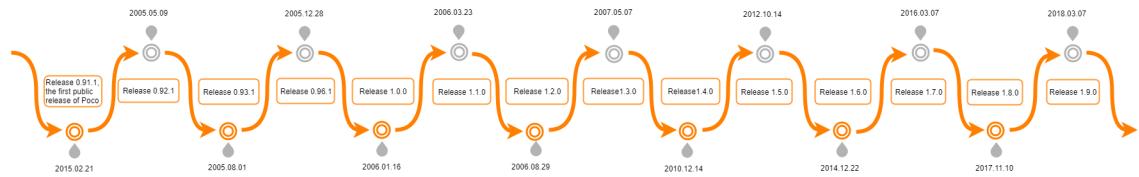
Figure 17.7: Timeline of the main releases *POCO*

The release process is based on the git branching model. A feature branch is branched off from the develop branch that contains the latest changes for the next release and will be merged back into the same branch once the new features are essential. Coordination and developments between different branches is tracked by the TSC. When the develop branch is near the desired state, all the features for the upcoming release are merged into the develop branch. A new release branch named by version number is branched off. This release branch is created from said develop branch. This branch exists for a long time during which the bugs can be resolved. The release branch finally will be merged into master branch when the release branch is deemed stable.

17.6 Functional View

This section describes the functional viewpoint. The functional view describes POCO's runtime functional elements and their responsibilities, interfaces, and primary interactions.

POCO provides multiple functionalities. To keep the functional view coherent, we will on only focus on the networking functionality. We will discuss the capabilities of the POCO, the external interfaces, and internal structure in that order. At last, we will present the structure model combining all what we identified in one figure.

17.6.1 Capabilities

Functional capabilities define what the project is required to do, and what is not required to do.

At the lowest level, the Net library contains socket objects, supporting TCP streams, server sockets, UDP sockets, multicast sockets, Internet Control Message Protocol, and raw sockets. Based on the socket objects, POCO provides building two TCP server frameworks, one as multi-threaded servers, and one for servers based on the Acceptor-Reactor pattern. The multi-threaded class and its supporting framework are the foundation for POCO's HTTP server implementation.

From a client side of view, the Net library provides objects for talking to HTTP servers, sending and receiving files, sending mail messages and receiving mails. The list below summarises some of the key objects of the `Net` library, in order to provide the described functionalities.

- `POCO_Socket` provides all comparison operators
- `POCO_DatagramSocket` provides an interface to UDP stream
- `POCO_StreamSocket` provides an interface to a TCP stream
- `POCO_ServerSocket` create a server socket, and provides an TCP socket interface
- `POCO_TCPServer` provides a multi-threaded TCP server that uses a server socket to listen for incoming connections
- `POCO_TCPServerConection` provides an abstract base class for TCP server connections
- `POCO_UDPServer`provides asynchronously polling of data which arrival is delegated
- `POCO_UDPClient` provides the functions to either send or receive UDP packets
- `POCO_HTTPServer` a sub object of `TCPServer` that provides a full-featured multi-threaded HTTP server
- `POCO_SMTPClientSession` provides a SMTP client for sending e-mail messages.

17.6.2 Design philosophy

The philosophy behind the creation of POCO can be captured by a quote from the main implementer of C++, Bjarne Stroustrup:

> Without a good library, most interesting tasks are hard to do in C++; but given a good library, almost any task can be made easy.

POCO is an open source, cross platform software project written in ANSI/ISO standard C++ and based on C++ Standard Library. As a collection of C++ class libraries, POCO tries to be conceptually similar to the Java Class Library, the .Net Framework or Apple's Cocoa.

17.6.3   External Interfaces

External interfaces are the date, event, and control flows between the function and others. POCO provides high and low level communication protocols. The purpose of the protocols is to establish a communication between the "Internet" and the internal procedures of a system. The external interfaces POCO provides can be identified as HTTP(S), SMTP, (S)FTP, TCP, and UDP. Additionally, POCO provides two more external interfaces: ICMP, and raw sockets which is named RAW. ### Internal Structure

To meet the requirement, a project can be design in a number of different ways. We will discuss the main element and their responsibilities at this section. Interfaces can be observed on a lower level by looking at the programming language, however, to not confuse with the external interfaces, we wont use this term here. Instead, those relations will be combined with the other connections between the elements.

The internal elements in POCO could be identified as `Web`, `Mail` and `File transfer`. The `Web` support two types of connections for server and client sessions, where both can use the `HTTP(s)`,`TCP` or `UDP` protocols. The `Mail` service provides a client to communicate with mail servers using the `SMTP` protocol. The `File transfer` service provides a client using the `(s)FTP` protocol. The `Socket` element is used to establish the communication with all other sockets in order to set and get various socket options such as timeouts, buffer sizes, and reuse address flags. In case the developed application needs secure communication with the external interfaces, they could make use of `OpenSSL` in order to encrypt the channel.

17.6.4   Structure Model

From what we have identified, we now can represent everything in a model, displaying the external interfaces and internal structure. There are many ways to create such a model, like by using a UML. We instead favour a box-and-lines diagram. Such a diagram doesn't require extensive knowledge of the UML notations, and doesn't restrict use to showcase information which might by hard or impossible to show in a UML.

Here all our elements, interfaces and connections can be found in one figure. Between each element, there is a connection. We used the arrows to showcase the data flow. As can be expected for all cased this goes both ways. The interfaces are show in orange, and the layer between the internal structure and the Internet has been shown by the dotted orange boundary. Note that here the client and servers are encapsulated by the protocols we recognised as interfaces. To distinguish the security part, it has been coloured grey for both the borders of the element boxes, and the connections.

17.7   Technical Debt

This section discusses the technical debt found in POCO. First the technical debt is identified. This section is followed by the historical view of the project, which describes the evolution of the project and the historical debt analysis over a period of one year. Next we will analyse the testing debt by describing the code coverage. Then we move on to a description of the impact from the identified technical debt, testing debt, and the historical view analysis. Finally, possible solutions will be described to reduce the technical debt of POCO.
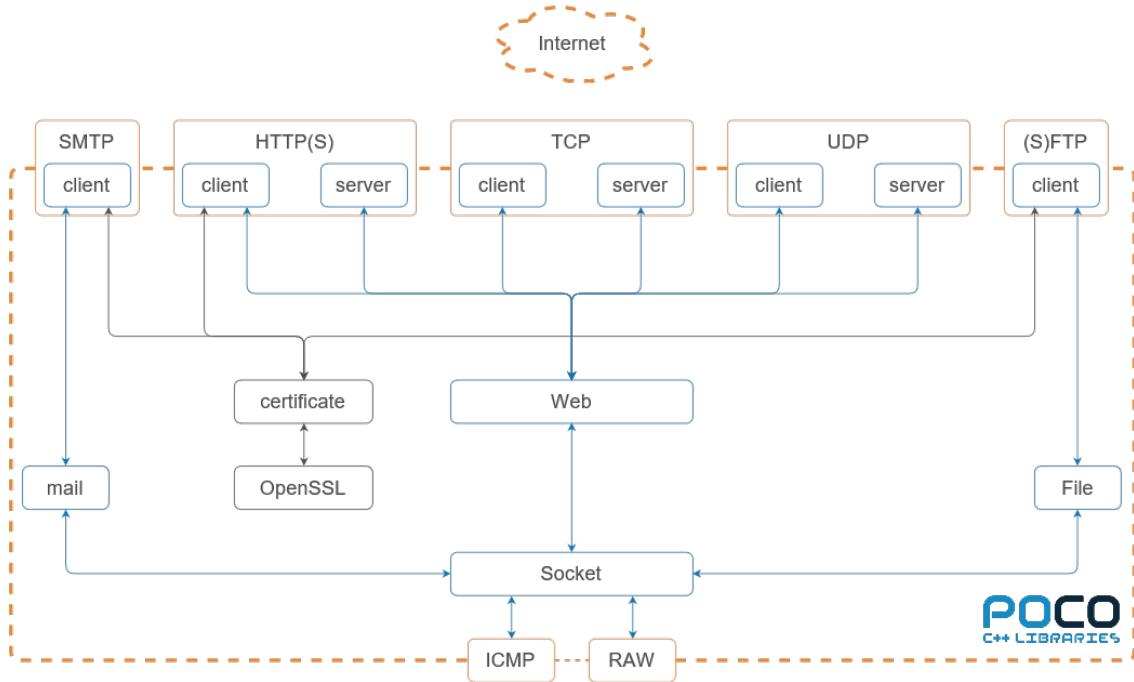
Figure 17.8: The functional view structure model

17.7.1 Identification

To assess the technical debt of POCO we used SonarCloud. SonarCloud is a tool that analyses the technical debt by detecting code smells. Detected smells are displayed below as the time it will take to resolve them.

Most of the code smells in the graph require approximately one day and thirteen hours to be resolved. The majority of the files contain few to non code smells. However, some outlier files can be noticed quite easily.

The file with the most smells turned out to be `NamedTuple.h`. This file implements tuples for a ranged size of 20 elements. Each size of the tuple has its own distinct, yet similar implementations. We found that 69% of the definitions are duplicates. The number of duplicated blocks is 7804 (14.1% of total blocks).

Another outlier contains a lot of code lines, while containing relatively few code smells. This file is the `SQLite3` source file. After looking at the commit history and the source file itself, we found out that the source code had been integrated from the SQLite distribution. To ease the integration, this file is a concatenation of all source files.

According to SonarCloud, POCO has a very low number of bugs and vulnerabilities. Many of the bugs are detected in JavaScript implementations.

SonarCloud will give a project a rating based on the SQALE. With a technical debt ratio of 0.4 %, POCO has been rated with an A.
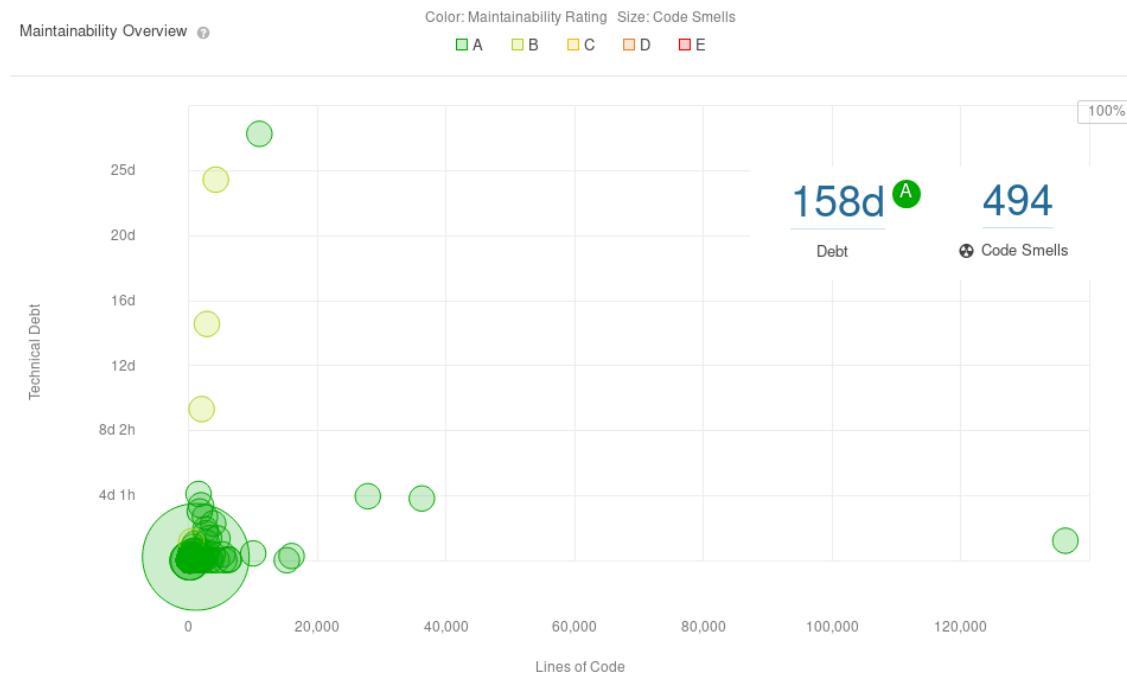
Figure 17.9: Maintainability overview of *POCO*

Figure 17.10: Bugs overview of *POCO*

17.7.2 Historical debt

Since the beginning of the project, the main functionalities of the `Net` and `SQL` modules were included with approximately 200 thousand additions. This trend remains almost the same for the earlier releases of the software until version 1.4.4 was released in July 2012. For this version major changes were made in the `Net` and `Foundation` libraries. The number of additions included in this change are more than 800 thousand, resolving a number of issues related to `IPv6` and securing sockets for data streaming. The change also added new functionalities and improvements in the `SQL` module. Moreover, a release happened in 2018 with approximately 250 thousand additions adding feature improvements for the `PosgreSQL` and `cmake` modules.

The graph below represents the evolution of the POCO project in terms of code additions and removal for the whole life cycle of the project:

Figure 17.11: Evolution of *POCO*

Another method to analyse the historical debt is by checking the code for certain keywords which could indicate a potential failure that need to be resolved, or a part that needs to be extended. The `TODO` keyword is used to indicate that a part of the code is incomplete and needs to be extended, while the `FIXME` keyword is used to indicate that a failure is discovered and needs to be resolved.

Between 2018 and 2019, the number of those two comments for the master branch remains almost the same. The POCO project version 1.9.0 release contains 97 comments marked as `TODO`, and 17 marked as `FIXME`. However, almost all `FIXME` comments are located in the `testsuite` modules. Moreover, many of those

comments are identified in the SQLite core file namely `sqlite3.c`. An example of a `FIXME` comment can be seen in Appendix D.

As a conclusion of the historical debt, the software continues to change with requests for fixes and enhancements and takes increasingly more time for contributors which impact the changeability and sensibility of the project.

17.7.3   Testing debt

The POCO project uses unit testing and Continuous Integration (CI) to maintain the project. Travic CI is used to build the project using multiple compilers, while AppVeyor is used to run the unit tests for different platforms. To maintain the code style and other practices applied in the project, POCO uses CII Best Practices to determine if they meet the criteria.

Although the POCO project does make use of unit testing and CI, there are no guidelines considering writing tests, and thus to prevent testing debt. In many cases, CI checks are bypassed by one of the assessors, and a pull request is merged even if some CI checks don't succeed.

The POCO project uses the CppUnit unit testing framework module to write down its tests. Although the POCO project uses unit testing to maintain the project, it doesn't make use of source code coverage analysis. However, by making some alterations during the compilation of the source code, the coverage information can be retrieved and the analysis can be performed. The source code coverage can be retrieved by using the Gcov tool.

We used Gcov to perform our own code coverage analysis. In total, only 52% is covered by unit tests. At Appendix C, the code coverage has been stated for each module.

The modules in the POCO project aren't equally treated; some modules have higher code coverage than other modules. Whether a module is well tested is debatable, however, we consider 70-80% to be the minimum acceptable code coverage. Looking at the code coverage per module, only four of the POCO project modules meet this criteria: `Foundation`, `JSON`, `Util`, and `Zip`. With 39973, 10243, and 10115 lines of code respectively, the `SQLite` , `ODBC`, and `Net` modules are remarkably poorly tested with code coverage of 3.26%, 37.28%, and 3.31%. Those three modules contribute 70% to the total lines of code.

17.7.4   Impact

The number of detected code smells from SonarCloud, as mentioned in the Identification section, don't necessarily indicate a malfunctioning build. Some code smells found by SonarCloud are related to duplicated blocks in the core libraries of POCO. The `Foundation` library has a total of 1732 duplicated blocks. Most of them are related to the tuple header file. Duplicated blocks have a negative effect on the code maintainability and safety by allowing a possible vulnerability to continue to exist in the copied code, while the developer is not aware of the copy.

From a historical point of view, the changes over the years have resulted in the decrease of reliability of the POCO project. For instance, a problem had been found five months after the release of version 1.4.4, and currently at the moment of writing, there are still open issues related to this change.

Overall, POCO was becoming more and more diverse by introducing new features to the libraries. This resulted in the increase of the number of duplication blocks in the code, making it more complex to maintain.

In addition, we noticed a lack of documentation after the release of version 1.7.8, which will impacts the project negatively for future modifications.

Whether the low code coverage is an issue depends on the effect of a failure. Multiple formal standards express the required minimal acceptable code coverage as a function of the critical level. POCO only provides libraries to other developers, and thus, the critical level depends on the usage of the POCO project. POCO could however increase their code coverage to lure in developers which need their features for high critical systems.

From the identified code smells by SonarCloud, the historical debt, and the code coverage, it appears that POCO only takes minimal effort to reduce the technical debt.

17.7.5   Solutions

The files with a lot of code lines are from external sources (e.g. `prettify.js` and `sqlite3.c`). Being external files explains why the test code coverage is extremely low. Developers of POCO assume that the external code is well tested. These external files are the result of concatenating multiple source files. As newer versions are released regularly, discarding the useless code every release is not viable. With regards to technical debt, we argue not to alter the external files. Instead, the technical debt should be explicitly documented. These files could also be stored in special locations. Currently the files are mixed with other source files. These strategies will not lower the technical debt, but will prevent raising false concerns.

Considering the files that contain a lot of duplicated blocks (e.g. `NamedTuple.h` and `Tuple.h`), re-evaluation the implementation seems to be the proper way to tackle this issue. Considering each size of a tuple has its own constructor, but share the same functions. Inheritance could be the key to reduce the number of duplicated blocks. Refactoring the implementation of the tuple all together might also be a solution as the current implementation can't extend the tuple beyond 20 items without altering code.

17.8   Conclusion

This report has described several aspects as described in Rozanski and Woods, centered around the POCO project. The aspects of POCO were divided into six sections. The first section described the stakeholders of POCO and their respective stake in the project. The stakeholder's interest and power were illustrated in a power/interest grid. The second section described the context view. The context view investigated the relationships, dependencies and interactions between POCO and it's environment. The context view was illustrated via a context model. This chapter also stated the important requirements and boundaries of POCO. The third section described the results of a small pull request analysis. 10 accepted and 10 rejected pull requests were analysed. Communication patterns that regularly occured were discussed in this chapter. The fourth section described the development view. The development view investigated the architecture that supports the development of POCO. It investigated the module structure of POCO and particullarly the dependencies of the modules in the Foundation Library. This chapter also presented a common design model which described the design constraints in order to maximise commonality and a codeline model of POCO which described the organisation of the source code and the protocol to build, test and release the system. The fifth section described the functional view. This chapter defined the architectural elements that give POCO it's functionality. Due to the large size of the library, this chapter mainly focussed on the NET library which is used to write network based applications. The last section described the technical debt

of POCO. Various methods have been used to determine the technical debt, the technical debt had been analysed and the methodology of technical debt supression in POCO had been investigated.

17.9 Appendices

17.9.1   Appendix A - The Developer Stakeholders

Contribution based on commit requests:

Place	Name	Commits
1	Aleksandar Fabijanic	1257
2	Günter Obiltschnig	1125
3	Francis Andre	720
4	Franky Braem	183
5	Pascal Bach	58

Contribution based on code distribution:

Place	Name	Insertions	Deletions
1	Aleksandar Fabijanic	2,681,625	2,768,735
2	Günter Obiltschnig	2,060,168	1,162,428
3	Francis Andre	62,394	48,386
4	Daniel Rosser	34,023	8,001
5	Eran Hammer	26,468	102

Contribution based on pull requests:

Place	Name	Pull Requests
1	Francis Andre	187
2	Aleksandar Fabijanic	32
3	Günter Obiltschnig	15
4	Roger Meier	11
5	Daniel Rosser	10

17.9.2   Appendix B - The Tester Stakeholders

Contribution based on code distribution to the unit test folder:

Place	Name	Insertions	Deletions
1	Aleksandar Fabijanic	34832	19923

Place	Name	Insertions	Deletions
2	Günter Obiltschnig	19515	7965
3	Marian Osborne	2948	1011
4	Francis Andre	2169	2122
5	Eran Hammer	1034	0

17.9.3   Appendix C - The Code Line Coverage

Code line coverage found by Gcov:

Module	Lines Covered	Total Lines	Percentage
Crypto	1912	2870	66.62%
Encodings	171	307	55.70%
Foundation	48717	63348	76.90%
JSON	2519	3218	78.28%
Net	335	10115	3.31%
NetSSL/OpenSSL	171	3556	4.81%
Redis	580	2141	27.09%
MySQL	199	3261	6.10%
ODBC	334	10243	3.26%
PostGreSQL	139	3782	3.68%
SQLite	14903	39973	37.28%
SQL	4409	6608	66.72%
Util	3867	4659	83.00%
XML	6735	10938	61.57%
Zip	1834	2180	84.13%

17.9.4   Appendix D - FIXME

An example of a `FIXME` comment.

```
if( p ){
  pFd->nFetchOut--;
}else{
  /* FIXME:  If Windows truly always prevents truncating or deleting a
  ** file while a mapping is held, then the following winUnmapfile() call
  ** is unnecessary can be omitted - potentially improving
  ** performance.  */
  winUnmapfile(pFd);
}
```

As can be understand from the `FIXME` comment, the function call to `winUnmapfile` depends on a third party vendor and omitting it could increase the performance.

Chapter 18

Polymer

Max de Krieger, David van der Leij, Sharon Grundmann, and Thomas Kolenbrander

18.1   Table of Contents

18.2   Introduction

Created by Google Inc. in 2015, Polymer is an open-source JavaScript library for building web applications. Through the use of web features like web components, HTTP/2 and service workers, Polymer allows its users to build modern web applications with minimal overhead and maximal performance. In addition to this, the team is dedicated to advocating for emerging standards to improve web development.

In this chapter, we analyze the architecture of Polymer according to concepts presented by Nick Rozanski and Eoin Woods [1]. We begin by analyzing Polymer's stakeholders. We identify major stakeholders associated with the project and their influence on decision-making processes and eventual success of Polymer. From there, we present two architectural views introduced by Rozanksi and Woods. The context view describes relationships, dependencies, and interactions between Polymer and its environment while the development view addresses aspects related to the system development process. Additionally, we analyze the technical debt of Polymer and discuss how variability is managed.

---

[1]Nick Rozanski and Eoin Woods. 2011. Software systems architecture: Working with stakeholders using viewpoints and perspectives. Addison-Wesley.

18.3   Stakeholder Analysis

We present the stakeholders of the Polymer project according to the stakeholder classes defined by Rozanski and Woods [2]. In addition, we identify other stakeholder classes that are relevant to this project.

18.3.1   Acquirer

The acquirer of Polymer is Google as they oversee the project with their team.

18.3.2   Assessors

Besides being the acquirer, Google can also be seen as the assessor of the project. Legal complications of the project are covered by a Contributor License Agreement (CLA) stating that contributors do not own any of the code they provided. Furthermore, the Polymer team also decides what standards to support, as stated in a blog post [3].

18.3.3   Communicators

The communicators consist of both the Polymer team and the Github contributors. External communication like contact with other stakeholders and marketing is done by the Polymer team while internal communication such as code reviews and documentation is done by both the Polymer team and Github contributors.

18.3.4   Developers

Polymer is developed by the Polymer team and Github contributors from the open-source community.

18.3.5   Maintainers

All developers can be considered maintainers as well. However, the Github contributors do not have any say in the future of Polymer as decisions like releases and roadmaps are made by the Polymer team. The members of the Google Polymer team that integrate new content (merge or close pull requests) are Steve Orvell (lead engineer, project coordinator) [4], Kevin Schaaf (lead engineer, project coordinator) [5], and Daniel Freedman [6].

---

[2]Nick Rozanski and Eoin Woods. 2011. Software systems architecture: Working with stakeholders using viewpoints and perspectives. Addison-Wesley.

[3]Gray Norton. 2018. Roadmap update, part 1: 3.0 and beyond updates on the polymer 3.0 release and what comes next. Retrieved February 28, 2019 from https://www.polymer- project.org/blog/2018-05-02-roadmap-update

[4]GitHub. Sorvell (steve orvell). Retrieved February 28, 2019 from https://github.com/sorvell

[5]GitHub. Kevinpschaaf (kevin schaaf). Retrieved February 28, 2019 from https://github.com/k evinpschaaf

[6]GitHub. Azakus (daniel freedman). Retrieved February 28, 2019 from https://github.com/aza kus

18.3.6 Suppliers

As a development and communication platform, Polymer makes use of Github. For distribution, the project relies on the npm package manager [7]. As the Polymer core team is paid and maintained by Google, Google can be seen as a supplier of the project as well.

18.3.7 Support staff

The actual staff consists of the Polymer core team and Github contributors. Communication channels between developers and users are available on Github, Twitter, Slack and Google Groups.

18.3.8 Testers

As all code committed should be tested, the developers (the Polymer team and Github contributors) are testers in the classical sense: They write tests for their code. Besides the developers, the users (mentioned below) can be seen as testers as well. They have the ability to report bugs on the Github page of Polymer.

18.3.9 Users

Polymer is used in many applications world-wide and in many of Google's products as well. This includes YouTube, Google Earth and Google Music. Besides Google, other notable users are: Netflix, ING, Coca-Cola, McDonald's and IBM [8].

18.3.10 Regulatory structures

The World Wide Web Consortium (W3C) is a regulatory stakeholder of the Polymer project. As the Polymer project directly involves creating and using web components, it is necessary that the team adheres to the web development standards maintained by W3C.

18.3.11 Competitors

Another group of stakeholders are competitors. Although they are not directly involved with the project and its management, they have an interest in it and can indirectly influence the design of products and success of the project as a whole. Some competitors of the Polymer Project include Angular [9], Vue.js [10] and React [11].

---

[7]Npm. npm. Retrieved April 14, 2019 from https://www.npmjs.com/

[8]Tim van der Lippe. 2018. Who's using polymer? Retrieved February 28, 2019 from https://www.polymer-project.org/blog/2018-05-02-roadmap-update

[9]Angular. Angular. Retrieved February 28, 2019 from https://angular.io/

[10]Evan You. Vue.js. Retrieved February 28, 2019 from https://vuejs.org/

[11]Facebook Inc. React – a javascript library for building user interfaces. Retrieved February 28, 2019 from https://reactjs.org/

18.3.12   Stakeholder influence

Using a power/interest grid, we categorize the stakeholders based on their influence and interest in Polymer as can be seen in the figure below. In the first place, Google and members of the Polymer core team have to be managed closely because they have the greatest influence on the success of the system in terms of funding and development. The next group with high power but low interest is the group of assessors at Google who ensure that the system conforms to standards and legal regulations. Aside from these groups, it is important to keep the contributors of the system including the GitHub community and other Polymer employees and users informed. Although these groups are highly interested in Polymer, they do not have much power. Lastly, the competitors of Polymer only have to be monitored since they have the least interest and influence on the system.
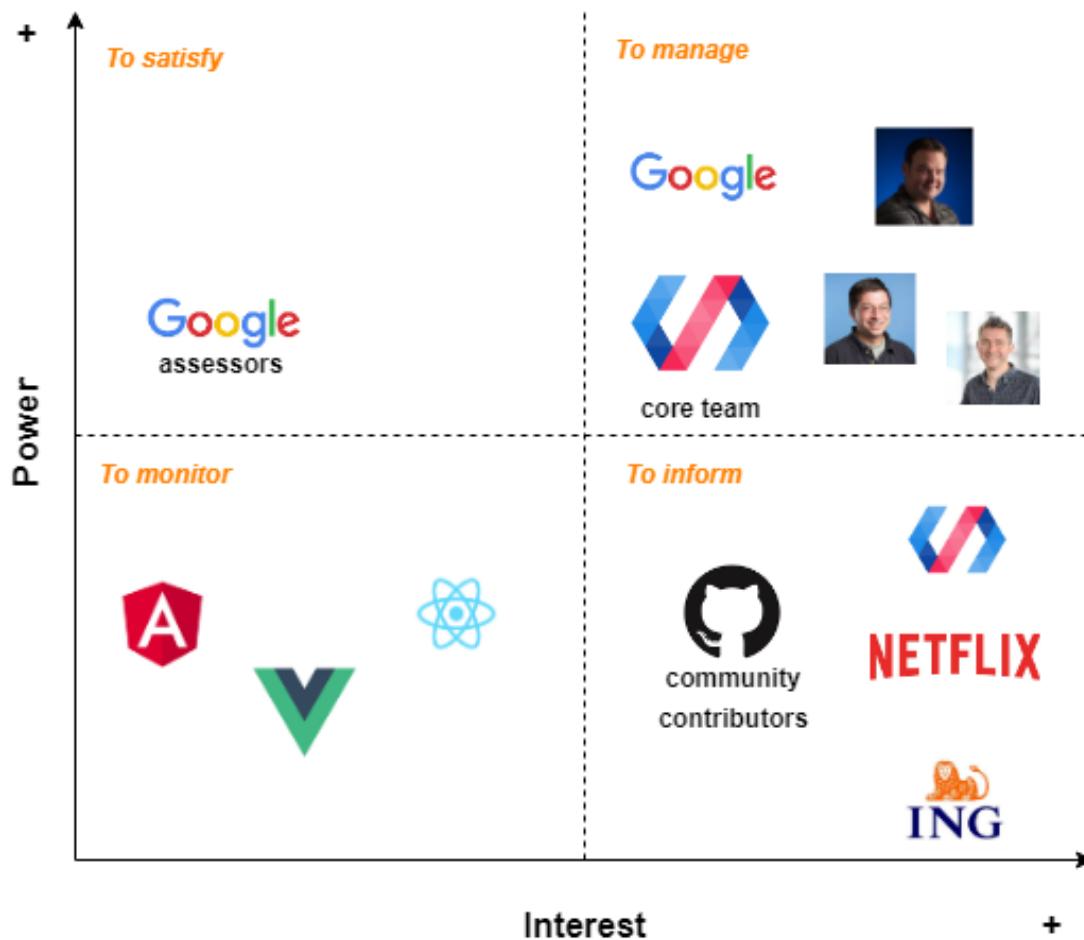
Figure 18.1: Diagram showing power and interests of stakeholders

18.3.13 Decision making process

Using the most commented pull requests from the Polymer project, insight might be gained regarding the decision making process of the Polymer team. Unfortunately, there is not much discussion about the impact of pull requests in the comments. Most of the comments are technical questions or clarifications. This could be due to the fact that most of the pull requests are created by Google employees, who are most likely on the same page with respect to the short and long term goals of the Polymer project. The code is extensively reviewed by one or more Google employees and discussed with the creator of the pull request. The most prevalent reason for the rejection of a pull request is that it is not tested well enough. Further analyses of popular pull requests are presented in the appendix.

18.4 Context View

In the context view, we consider the system as a black box while describing its main functionalities and relations with its external environment.

18.4.1 Key functional requirements

In order to identify the key functional requirements we first examine the feature overview [12]. The main features of Polymer identified include:

- Custom elements that function like DOM elements
- Elements consisting of ECMAScript modules with classes
- Elements have an encapsulated DOM that is populated by Polymer
- Declarative syntax to attach listeners to elements children, with gesture support
- Data binding of properties and attributes with observers and computed properties

The underlying goal of these requirements is to "help developers unlock the web's full potential" [13].

18.4.2 External entities

We have identified the following external entities:

- **Developers** - The Polymer library is developed by a core team at Google and contributors from the open source community on Github.
- **Users** - Polymer is used by a large number of companies including well-known companies like: Netflix, ING, Coca-Cola, McDonald's and IBM. Besides these companies Polymer is also used by its main developer, Google, and can be found in various products created by Google.
- **Runs in** - Polymer is used to create web applications and thus requires a browser to work. Thus, Polymer is dependent on the features that certain browsers support.
- **Competitors** - Competitors of Polymer include other Javascript libraries focused on web applications. At the time of writing, popular alternatives to Polymer are Vue.js, AngularJS and React.

---

[12]The Polymer Project Authors. 2018. Polymer library - polymer project. Retrieved February 28, 2019 from https://polymer-library.polymer-project.org/3.0/docs/devguide/feature-overview

[13]Polymer. Polymer project. Retrieved April 12, 2019 from https://www.polymer-project.org/

- **Development tools** - Polymer is developed in JavaScript and is distributed through the npm package manager. Github is used to support the development of Polymer. The included testing framework runs on Node.js and includes several JavaScript frameworks. The JavaScript frameworks included in the testing environment are: Mocha, Chai, Async, Lodash and Sinon [14]. For continuous integration purposes Travis CI is used.
- **Communication** - The official communication channels for Polymer are their website, Github, Twitter, Slack and Google Groups.
- **License** - The Polymer library is available under the BSD license.

18.4.3   External interfaces

Because Polymer is a development library, the external entities that use it as a development library do not request data from it but interact with it as a tool rather than a provider. Therefore the external entities only interface with Polymer through development tools which can be summarized as follows:

- Polymer components
- Web component tester
- Polymer CLI (command line interface)

The Polymer components let developers create web components and are therefore the main interface of the project.

The web component tester is a tool that allows developers to test these components using scripts in an automated way. This tool, in turn, interfaces with the previously mentioned test frameworks. Since these frameworks are included within the testing tool, Polymer does not depend on updates of these frameworks being compatible.

Lastly, the polymer CLI allows developers to execute a myriad of project structure tasks such as running a local web server, creating a project with the correct structure and creating Polymer components from templates.

For Polymer to work properly, developers need to make Polymer interface with npm to retrieve the required packages. Next to this, the framework itself is also hosted on the npm repository. This interaction is done via the HTTP protocol.

Although the volume of requests for this interaction is relatively small (only for package updates and setting up the project), it can pose a problem for external entities that use Polymer if the service is down for an extended amount of time. The service level of npm however seems to be near a 100% uptime [15].

18.4.4   Context view diagram

The context view diagram in the figure above gives an overview of the different entities involved in Polymer and their relationships. Polymer runs in different browsers, which all have the ability to support or exclude certain standards. The developers of Polymer have take into account these decisions. The development

---

[14]Polymer.   Tools/packages/web-component-tester at master polymer/tools.   Retrieved February 28, 2019 from https://github.com/Polymer/tools/tree/master/packages/web-component-tester
[15]Npm http service uptime – service uptime in the past 90 days. Retrieved February 28, 2019 from http://ping.npmjs.com/
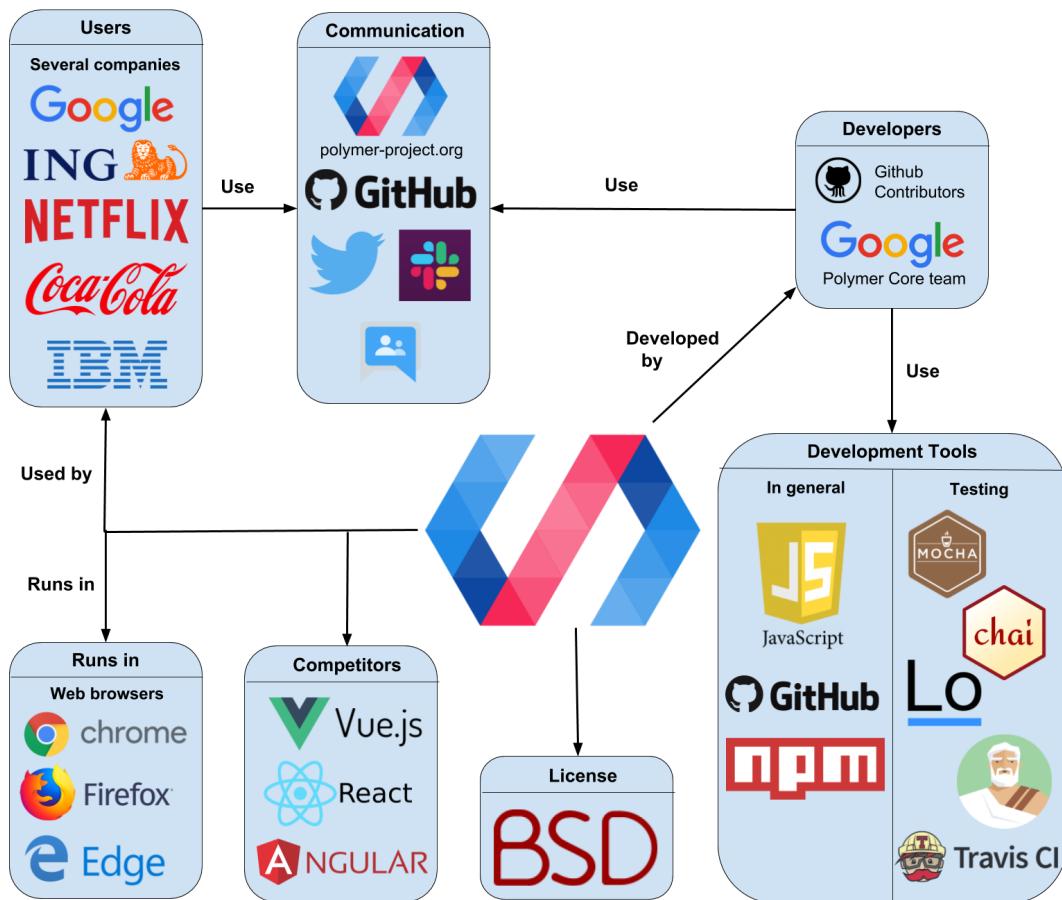
Figure 18.2: The context view for Polymer, showing the entities and their relationships

process is supported by several tools/platforms such as npm and Travis CI. Developers and users of Polymer are able to communicate with each other through several platforms including Github, Twitter and Slack.

In general, the diagram shows that Google is one of the most involved parties, as they can be found in five different groups: Users, Developers, Web browsers (Google Chrome), Communication (Google Groups) and even Competitors (Angular).

18.5   Development view

The development view describes the architecture that supports the software development process. In the paragraphs below, the codeline model, source code structure, code modules, code dependencies and the common design model are analyzed.

18.5.1   Codeline model

The codeline model is in place to guide the development process. It explains how the code is controlled, how it should be maintained and the automated tools used to deploy the software.

18.5.1.1   Build and Test Approach

All Polymer projects make use of Polymer CLI [16], a command-line tool for the installation, testing and building of Polymer projects and Web Components. There is no configuration required to use Polymer CLI which makes creating a Polymer project rather easy. The tests are run using the so-called web-component-tester (WCT) [17], which is a browser-based testing environment created by the Polymer team. This environment also supports SauceLabs [18] which allows for multi-platform cloud-based testing.

18.5.1.2   Release Process

Releases are done via the Github releases [19] and tags [20] sections. In this section, Polymer is released for both the 2.x and 3.x versions of the library. Every release lists the new features, meaningful changes and a raw list of all commits in this release. The release itself contains a tar and zip archive with the source code. To our knowledge there seems to be no documentation on what constitutes a release candidate and it seems a release is pushed when a significant amount of commits are made since the last release and the continuous integration does not raise an error.

---

[16]Polymer.   Tools/packages/cli   at   master   -   polymer/tools.   Retrieved   March   21,   2019   from https://github.com/Polymer/tools/tree/master/packages/cli

[17]Polymer.   Tools/packages/web-component-tester   at   master   polymer/tools.   Retrieved   February   28,   2019   from https://github.com/Polymer/tools/tree/master/packages/web-component-tester

[18]SauceLabs. Cross browser testing. Retrieved March 21, 2019 from https://saucelabs.com/

[19]Polymer. Releases - polymer/polymer. Retrieved March 21, 2019 from https://github.com/Pol ymer/polymer/releases

[20]Polymer. Tags - polymer/polymer. Retrieved March 21, 2019 from https://github.com/Polym er/polymer/tags

18.5.1.3 Continuous integration

After every commit the continuous integration tool TravisCI [21] is invoked. This utility executes a build script [22] which performs the following tasks:

1. Specifies variables such as the Linux distribution and NodeJS version
2. Installs and builds the project and dependencies using Gulp
3. Runs the ES linter on the codebase
4. Runs all tests on both Google Chrome and Mozilla Firefox

18.5.1.4 Version control

As mentioned previously in this chapter, the codebase of Polymer can be found on GitHub. The project uses Git for version control. This means that the repository has to be forked by the contributors including Polymer developers before they can make changes. Pull requests are thoroughly reviewed by the core Polymer team before they are accepted and merged or disregarded. These changes must comply to the standard design approaches.

18.5.2 Source code structure

In the figure below, a hierarchical view is shown of the file structure of the polymer project. It can be seen that the project is divided into three parts namely the actual library which is contained in lib, the testing part which is contained in the test folder and the root of the project where most configuration files are located. These configuration files specify dependencies or the continuous integration files for example. What is especially interesting is the size of the testing files in comparison with the actual framework itself. We can see that this part is about two to three times the size of the actual framework. This shows that the Polymer has a comprehensive test suite high on the list of priorities.

18.5.2.1 Lib

As seen in the figure below the actual framework itself is divided into 4 elements. These are the elements, utils, mixins and legacy folders. The utils folder contains utility classes such as async and array operations. The mixins folder contains a couple of base mixins. A mixin is a function that takes a class and returns a subclass. The legacy folder contains files that are necessary to stay compatible with older versions of Polymer. Lastly, the elements folder contains some base elements that allow users to easily use standard patterns and not have to redefine these patterns.

18.5.2.2 Testing

In the figure below the testing file structure is shown. It can be seen that the majority of the tests consist of unit tests. There are a small amount of performance tests. Lastly there are some smoke tests as well which run certain functionality in browser and check if the expected result is achieved.

---

[21]Travis CI. Travis ci - test and deploy your code with confidence. Retrieved March 21, 2019 from https://travis-ci.org/
[22]Polymer. Polymer/.travis.yml at master - polymer/polymer. Retrieved March 21, 2019 from https://github.com/Polymer/polymer/blob/master/.travis.yml
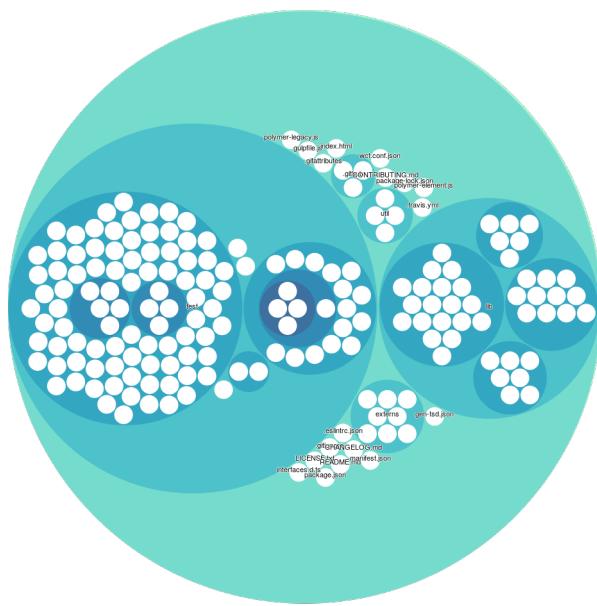
Figure 18.3: Source code file structure. The white circles are files. The names of the corresponding folders is displayed
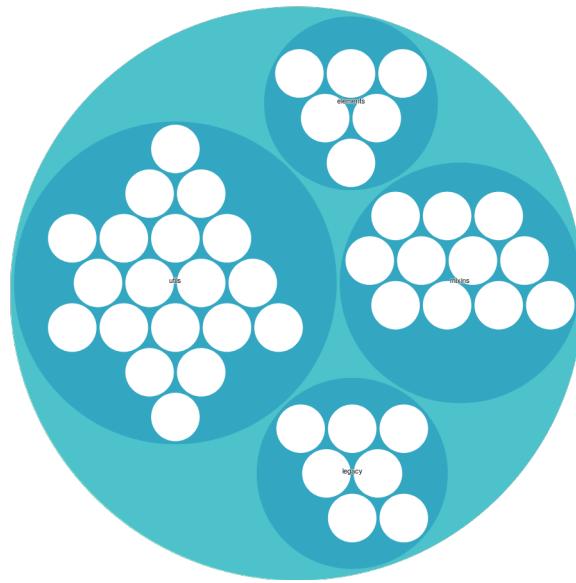
Figure 18.4: Library file structure
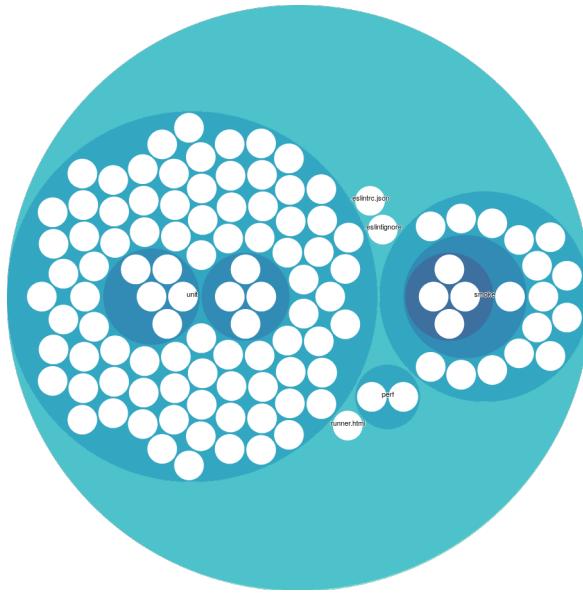
Figure 18.5: Testing file structure

18.5.3 Code modules and dependencies

A visualization of the source code modules can be found in the figure below. The source of the core Polymer API is transparent in the sense that every class in the project is something that contains a functionality to the Polymer users. This makes explaining the code modules straight forward, meaning that what you see in the source is what you get as a user.

Next to the main functionality the Polymer project offers, there are three additional folders with source code for different goals. The source code located in the *Externs* module contains only variable and function declarations. This is necessary when using the project in combination with the Google Closure Compiler [23]. The source code in the *Util* folder is very minimal and meant for assisting the developers of Polymer (with tools such as a changelog-generator, etc.) and the *Test* folder is, as expected, full of tests that ensure the quality of the Polymer project.

The features polymer offers are divided in four modules: **elements**, **mixins**, **utils** and **legacy**. The source code contained in **elements** could be classified as a code module that serves the purpose of providing the Polymer user with pre-defined HTML elements. All HTML elements are based on the *polymer-element.js* file in the root of the directory, the *polymer-element.js* is therefore a dependency. Other dependencies of this module are **mixins** and **utils**.

Mixins is a second module that provides Polymer users with classes that offer Polymer meta-programming features. Alternatively, mixins can be seen as interfaces that already have an implementation. This module depends on the **element** and **utils** modules.

The third module, **utils** is, as the name suggests, a collection of functions that provide utilities needed for many web component development tasks. This module is largely independent but contains a file that imports

---

[23]Google. Closure compiler | google developers. Retrieved March 20, 2019 from https://develope rs.google.com/closure/compiler
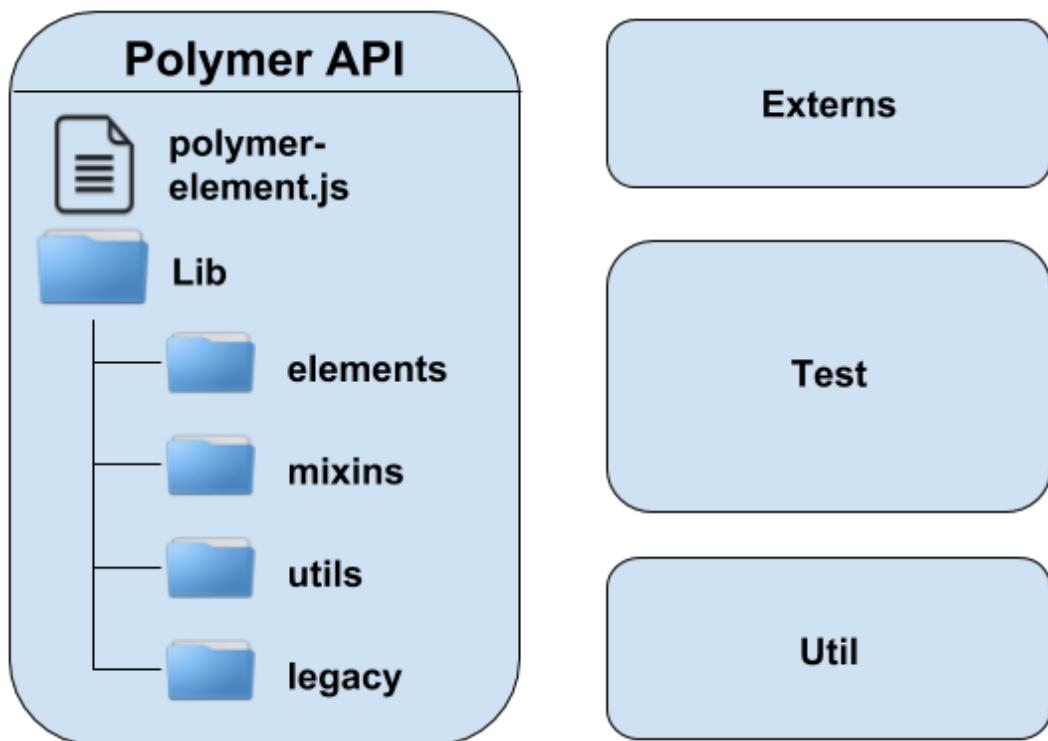
Figure 18.6: The code modules for Polymer

functionality from the **elements** module.

Lastly, the **legacy** module provides the utilities to be compatible with older versions of Polymer. These can be used while migrating web components to a newer version of Polymer for example. This module depends on the other modules **mixins** and **utils**.

18.5.4 Common Design Model

The common design model describes the standardization approach used by developers to ensure commonality across element implementations. This includes identifying the common processing required across elements, standard design approaches and standard software components used by Polymer. In order to construct an accurate model, we analyzed the Polymer GitHub repository and online documentation. Additionally, we consulted an open source contributor and later intern of Polymer, Tim van der Lippe (TimvdLippe on GitHub) about his experience working with the core team. His input together with our research results are presented in this section.

18.5.4.1 Standard design

Polymer standardizes its elements in order to make them usable and easy to maintain. In this regard, all elements must conform to the Polymer element style guide [24] which defines how to specify properties and documentation.

- **Coding style** - The Polymer team uses the Google JavaScript Style Guide [25] and Google HTML/CSS Style Guide [26] to structure their code. Adherence to these guidelines is mandatory and all contributions to the projects are checked for conformance. This ensures that the coding style of the project stays uniform and developers can easily understand and maintain the project.

- **Commit style** - Unlike other Google projects, Polymer's standard guide to contributing to the project is fairly flexible. The guide [27] states how bugs should be filed and pull requests can be made by the GitHub community. The team also has a template for reporting issues detected in the code.

18.5.4.2 Standard elements and common processing

Polymer primarily exposes one custom element to the user that is called **PolymerElement**. This element is built up from several mixins (described in the code modules section above). The pattern that can be seen in all of the source code is that functionality is built in mixin classes (or higher-order functions) and the actual exported elements are constructed using these mixin classes.

Next to **PolymerElement**, Polymer offers several other elements that the user can use such as the ArraySelector element [28]. These additional elements are extensions of **PolymerElement** with additional

---

[24]The Polymer Project Authors. Document your elements - polymer project. Retrieved March 21, 2019 from https://polymer-library.polymer-project.org/3.0/docs/tools/documentation

[25]Google. Google javascript style guide. Retrieved March 21, 2019 from https://google.github.io /styleguide/jsguide.html

[26]Google. Google html/css style guide. Retrieved March 21, 2019 from https://google.github.io /styleguide/htmlcssguide.html

[27]Polymer. Polymer/contributing.md at master - polymer/polymer. Retrieved March 21, 2019 from https://github.com/Polymer/polymer/blob/master/CONTRIBUTING.md

[28]GitHub. Polymer/array-selector.js at master polymer/polymer. Retrieved March 20, 2019 from https://github.com/Polymer/polymer/blob/master/lib/elements/array-selector.js

functionality. An example is shown below:

```
// The functionality is defined as a function that returns
// a class with the desired properties.
let ArraySelectorMixin = dedupingMixin(superClass => {
  ...
})

// The actual class to be exported is an instantiation of the PolymerElement
// with functionalities from the above defined functions.
let baseArraySelector = ArraySelectorMixin(PolymerElement);
class ArraySelector extends baseArraySelector {
  ...
}

export { ArraySelector }
```

Not all elements are defined in this way, for instance the element **CustomStyle** is simply defined as a class that extends the standard HTMLElement and then calls `window.customElements.define(...)` to expose this to the user.

18.6   Technical debt

In this section we discuss technical debt related to the Polymer project. Technical debt can be described as the gap between making a change work and making a change perfectly. We assess the technical debt by analyzing pull requests, measuring code quality and interviewing Tim van der Lippe, an ex-member of the Polymer core team.

18.6.1   Discussion in pull requests

After analyzing the thirty most recent pull requests that had more than five comments, the conclusion must be drawn that there is not much discussion about technical debt in the Polymer pull requests. Only in one single case (5451 [29]) was there a request to refactor a line of code to make it more reusable, which can be seen as a discussion about technical debt. One other pull request (5347 [30]) was on the topic of cleaning up code after a conversion. More often (four out of thirty pull requests), a change is requested to add one or two comments to improve readability.

18.6.2   Code quality

In order to evaluate the code quality of Polymer, we made use of two code evaluation tools: DeepScan [31] and SonarQube [32]. Both of these tools are static code analysis tools where DeepScan is specifically designed

---

[29]GitHub. 2018. Retrieved March 20, 2019 from https://github.com/Polymer/polymer/pull/5451

[30]GitHub. 2018. Retrieved March 20, 2019 from https://github.com/Polymer/polymer/pull/5347

[31]deepscan.io. How to ensure javascript code quality | deepscan. Retrieved March 20, 2019 from https://deepscan.io/home/

[32]SonarSource S. A. Continuous inspection | sonarqube. Retrieved March 20, 2019 from https: //www.sonarqube.org/

for Javascript. The evaluation of the codebase was performed on the 6th of March 2019, so results may have changed in the mean time.

18.6.2.1 DeepScan

The result of running DeepScan can be seen in the figure below. According to DeepScan, the overall grade of the project was 'Good' which is the highest grade available. The tool found a total of 12 issues, 9 of which are related to code quality and marked as 'low impact'. The remaining 3 issues are related to possible errors in the code and considered 'medium impact'. The issues do not seem to have a common quality and are all rather different, so there is no pattern to be found which means that the issues are not structural and rather incidents.
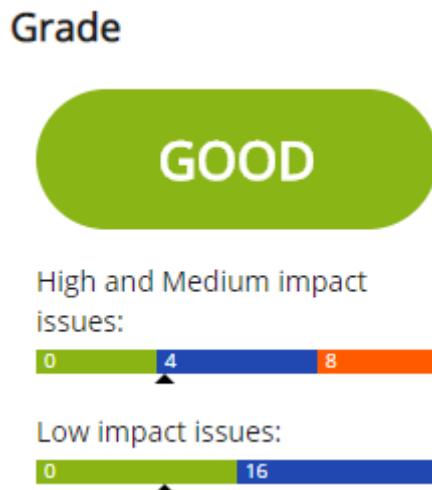
Figure 18.7: Evaluation results of DeepScan

18.6.2.2 SonarQube

SonarQube rates the code on several different aspects. The results can be found in the figure below. We evaluate four of them:

- **Reliability** - The initial reliability rating for Polymer is 'C' (on a scale from A to E) because SonarQube discovered 87 bugs of which 80 were classified as major. Upon further inspection it turned out these 80 bugs were all caused because the `<title>` tag is missing in the test HTML files. Because these files are just used for testing purposes, a title is not necessary and these bugs can therefore be discarded. The other 7 minor bugs are in the test HTML files as well. These bugs can be discarded in the same way, as they include missing "alt" attributes to images and some styling issues which do not matter for testing purposes. Since all bugs are not applicable to the project, we can conclude that Polymer should actually get an 'A' for reliability.
- **Security** - SonarQube was not able to find any vulnerabilities and the security is therefore graded with an 'A'.

- **Maintainability** - The maintainability is defined by the number of code smells detected. SonarQube was able to detect a total of 25 code smells. Over half of these smells are related to unused imports or variables. The other half consists of various issues. The rating for maintainability is also dependent on the size of the project, with a larger project code smells are more likely and thus to be expected. Due to the large size of the project and the small number of code smells Polymer received an 'A' rating for maintainability.
- **Duplication** - SonarQube does grade the duplication but just shows the results. In total, 13.1% of the lines was duplicated. However, all the duplications are located in the test folder which is no surprise as testing often requires repeating the same instructions with small differences. We can therefore conclude that duplication is no problem in the Polymer project.

Figure 18.8: Evaluation results of SonarQube

Both DeepScan and SonarQube report very positive results for the Polymer project. It can therefore be concluded that the project does not contain a lot of technical debt and is very healthy in general.

18.6.3   Evolution of technical debt

In an e-mail interview with Polymer ex-core team member Tim van der Lippe we discovered that the team suffered from technical debt due to non-finalized standards of the W3C foundations WebComponents. He mentioned however that as these standards were finalized, Polymer has eliminated this debt by adhering to these standards. In the same interview Tim mentioned that "We did do regular cleanups and published multiple versions to fix previous mistakes/usages of non-shipping standards". The evolution of technical debt was described as an organic process and by doing these cleanups the team was able to keep the amount of technical debt under control. Besides the non-finalized standards there were no major factors introducing technical debt in the project.

18.6.4   Testing debt

The web-component-tester (WCT) environment does not natively support the generation of code coverage. In order to generate code coverage an external plugin is required. The istanbul tool [33] has been made

---

[33]Istanbul. Istanbul, a javascript test coverage tool. Retrieved March 21, 2019 from https: //istanbul.js.org/

available for the WCT environment but is not maintained actively. Due to this we were unfortunately not able to generate test coverage reports.

The project in general does not rely on line coverage requirements but instead requires that all functionalities are tested. Unfortunately we were not able to gather the actual coverage statistics of the project. The large number of test cases show that testing is definitely taken into consideration when contributions are made. Testing is also often discussed in the pull requests which shows that testing is an integral part of the project. Because of the aforementioned reasons it would be obvious to assume that the testing debt is rather low. However, due to our lack of information we can not be sure of this. When inspecting the code manually we did manage to find some testing debt. An example of this is the "path.js" file in the utils package which has a testing class that does not cover all its functions. This shows that there is at least some form of testing debt present in Polymer.

18.7   Variability

Polymer is a web component library and needs to be compatible with all major browsers in order to be adopted by web developers. Naturally, major browsers differ in implementation and this introduces variability in Polymer. Most variability in Polymer is not concerned with providing different features to the developer but instead, the developer should take it into account in order to deliver consistent web components to all major browsers. Next to browser compatibility, Polymer also creates variability by introducing flags for certain commands in the Polymer command-line interface. Polymer makes use of load-time and run-time binding for its variability implementation, allowing for flexible development and reconfiguration. However, these mechanisms for load-time and run-time binding incur a memory and performance overhead, since all variations are compiled into a single binary and consistency conditions must be checked at run-time [34].

18.7.1   Examples and implementation

Below are examples of variability in Polymer and their implementation.

18.7.1.1   Polyfills

Some features used by Polymer are not supported by major browsers [35], polyfills are used to build a bridge between them. A polyfill is a term used to describe a library that checks if a browser API supports a certain functionality; if it does not, the polyfill will intercept any calls to that functionality and run its own implementation [36].

Implementation

Polymer uses the polyfills from webcomponents.org [37]. To load the polyfills to the browser, the Polymer user can load either `webcomponents-bundle.js` or `webcomponents-loader.js` from the `@webcomponents`

---

[34] Sven Apel, Don Batory, Christian Kastner, and Gunter Saake. 2016. Feature-oriented software product lines. Springer.

[35] The Polymer Project Authors. Browser support overview - polymer project. Retrieved April 2, 2019 from https://polymer-library.polymer-project.org/3.0/docs/browsers

[36] Axel Rauschmayer. 2014. Speaking javascript - an in-depth guide for programmers. Retrieved April 2, 2019 from http://speakingjs.com/es5/ch30.html

[37] webcomponents.org. Webcomponents.org. Retrieved April 2, 2019 from https://www.webcom ponents.org/

module [38]. The first script retrieves all polyfills and loads them to the browser, resulting in extra bytes sent over the network (even unnecessary polyfills are sent). The alternative is the latter script, which assesses what features are supported by the browsers and then only retrieves the relevant polyfills. The latter option results in possibly less bytes sent over the network but does require an extra round-trip. This variability is bound at load time when the appropriate JavaScript file is used.

18.7.1.2   Command-line interface

The Polymer project offers a command-line interface (CLI) that offers useful functionality to developers such as building the application [39]. The commands `polymer build` and `polymer serve` accept flags that provide functionality. Because the `polymer build` command has fourteen possible flags, certain presets are offered that bundle common flag combinations.

Implementation

The Polymer team created a TypeScript interface named `Command` which all CLI commands implement. At run-time, a user can use the configuration options provided by the Polymer CLI to configure his or her project. This includes defining the root of the project, the build files, shell and fragments that could be implemented. In a specific command such as `build`, the supported flags are described by creating an array of `ArgDescriptors`, which also is a TypeScript interface. During the execution of the command, the flags are implemented as conditionals. An example is the `--js-minify` flag which is implemented in the `js-transform.js` file as follows [40].

```
if (options.minify) {
  doBabel = true;
  // Minify last, so push first.
  presets.push(babelPresetMinify);
}
```

18.7.2   Partial feature model

In the figure below we can see a figure of the variability in Polymer. This model does not contain all the features of Polymer but rather the ones where a user (developer) can introduce variability. The figure is from the view of the developer since those are the users of Polymer. The browser choice is not explicitly given in Polymer but it does influence whether the use of polyfills, shady CSS and shady DOM can be enabled. For example: newer versions of Chrome do not require any of these features and if a user of Polymer decides to only support this browser these features are not used.

---

[38] The Polymer Project Authors. Polyfills - polymer project. Retrieved April 2, 2019 from https://polymer-library.polymer-project.org/3.0/docs/polyfills

[39] The Polymer Project Authors. Polymer cli commands - polymer project. Retrieved April 2, 2019 from https://polymer-library.polymer-project.org/3.0/docs/tools/polymer-cli-commands

[40] Polymer. 2018. Polymer-build/js-transform.ts at master - polymer/polymer-build - github. Retrieved April 2, 2019 from https://github.com/Polymer/polymer-build/blob/master/src/js- transf orm.ts
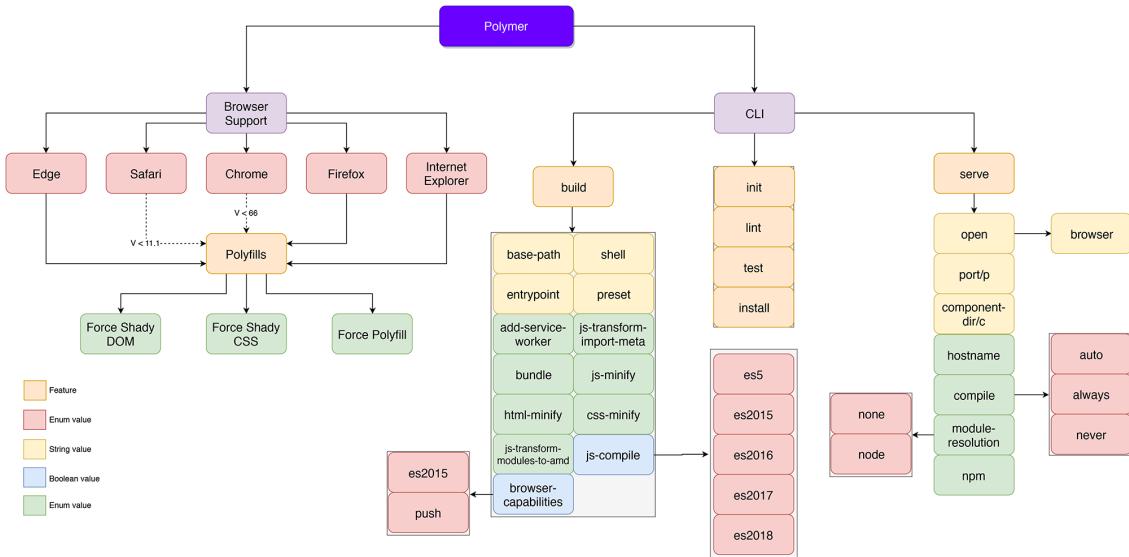
Figure 18.9: Variability feature model for Polymer

18.7.3 Variability management

How variability is managed is different for users and developers of the project. Here users are defined as 'people using Polymer to create a website' and developers as 'people contributing to the Polymer project'. The following paragraphs explain how variability is managed within the Polymer project.

18.7.3.1 Variability management for users

When building a project with Polymer, the user has to decide on hosting features and supported browsers [41]. The first decision is based on whether the hosting provider supports user agent detection and the ability to serve different files to different user agents. If that is the case then the user is able to deploy multiple builds and depending on the browser used, the most optimized build is served. If this is not supported by the hosting provider, the user has to generate a single build and has to decide which specific browsers to support. For an individual build Polymer supports 17 parameters [42], allowing the user a lot of freedom in their choices. Such an amount of parameters can be overwhelming and therefore three presets are offered in addition which each provide a different, standard level of support for browsers.

---

[41]Polymer. Build for production - polymer project. Retrieved from https://polymer-library.poly mer-project.org/3.0/docs/apps/build-for-production

[42]Polymer. Polymer.json specification - polymer project. Retrieved from https://polymer- library.polymer-project.org/3.0/docs/tools/polymer-json

18.7.3.2   Variability management for developers

For the support of older browsers, Polymer relies on polyfills created by the Web Components project [43]. So, developers of Polymer do not have to directly handle the support of older browsers. Since the project is heavily reliant on these polyfills, the Polymer team has also been concerned with the development and maintenance of these polyfills [44].

Whether Polymer is fully (natively) supported or not is mostly dependant on the choice of browsers to support the Web Components API. Developers themselves have little to no influence in this decision unfortunately.

18.8   Conclusion

In this chapter, we have analyzed Google's Polymer project using several architectural views. We analyzed its stakeholders and identified Google as the most influential. Google plays a vital role in the development, funding and decision making of the project while also being assessors and users of Polymer. In the context view, we saw that Polymer is heavily reliant on other systems and frameworks for communication, support and testing. The project is developed in JavaScript and distributed through the npm package manager with its key functionalities made available through the Polymer CLI, Polymer elements and web components. The development view presented the design and code structure of Polymer. It was interesting to note that the testing module of Polymer is almost three times as large as the library. As expected of any Google project, the design approach adopted by Polymer is based on Google's style guide. With regards to technical debt, our analysis shows that the project is very healthy. Most of the code smells detected were related to the testing module and could be seen as trivial. Just as Tim said, the project is very well tested and technical debt is handled efficiently. We also analyzed how variability is managed in Polymer given that it is a web-based project. The analysis shows that most of the variability has to do with compatibility with browsers and options made available to developers through the CLI.

Overall, analyzing Polymer was interesting. We are pleasantly surprised by the quality of code especially with regards to tests and the low technical debt maintained by the team. One area that can be improved is the documentation of the project. The documentation is somewhat lacking, making it difficult to assess how certain decisions are made by the team.

18.9   Appendix A - Analyses of pull requests

The analyzed pull requests below are the most commented pull requests from the whole Polymer repository. Labeling of the comments was done using the following codes:

- **AC** - Author (pull request creator) clarifies his actions in the pull request.
- **AT** - Author makes a technical comment regarding his pull request.
- **AG** - Author makes a comment regarding GitHub tasks: assigning of reviewers, asking to merge, announced a rebase against master, etc.

---

[43] webcomponents. Webcomponents/webcomponents.js: A suite of polyfills supporting the html web components specs. Retrieved from https://github.com/webcomponents/webcomponentsjs

[44] Polymer. Roadmap update, part 1: 3.0 and beyond - polymer project. Retrieved from https://www.polymer-project.org/blog/2018-05-02-roadmap-update

- **UF** - User of the library posts a comment regarding the impact of the pull request on existing features or issues. Mostly this indicates that a user is asking whether this pull request will fix his current problem.
- **UT** - User of the library asks a technical question about the changes made by the author.
- **GG** - Google employee makes a comment regarding GitHub tasks.
- **GF** - Google employee makes a comment regarding the impact of the pull request on the current state of the project.
- **GT** - Google employee makes a technical comment.

18.9.1   Approved - #2642

Created - 25-10-2015, merged - 04-12-2015, first included in Polymer v1.2.4

Pull request #2642 is listed as the most commented pull request of the entire project. Contributor nazar-pc fixed a problem concerning incorrect CSS selector specificity. Merging this pull requests fixed at least three listed issues at the time.

The first Polymer team member to respond was azakus and stated that he would attempt to do the merge of this large pull request. Later Polymer team member sorvell commented saying that the Polymer team was considering solving the issue before the PR was made, but stated that they were concerned about the impact on the performance and the Polymer team planned to handle custom CSS properties different in the long term. azakus reviewed the code and finally merged the pull request.

Label	#comments
AC	4
AT	5
AG	4
UF	4
UT	2
GG	7
GF	2
GT	0

18.9.2   Approved - #5000

Created - 19-12-2017, merged - 10-12-2018, first included in Polymer v2.4.0

Polymer contributor 43081j created pull request #5000 where he added return type annotations in function descriptions. This pull request would fix at least two issues where correct return types in function descriptions are necessary, as is the case in many scenarios where Polymer is combined with TypeScript.

The large amount of time merging took was partly due to the pull request only being mergable once pull requests of other Polymer projects got merged such as polymer-analyzer#821 and a Polymer tool that generates TypeScript declarations gen-typescript-declarations#56. TimvdLippe commented that this pull request would get Polymer close to having correct type annotations and continued to follow the updates. Once the pull request was mergable, azakus did so.

Label	#comments
AC	2
AT	7
AG	3
UF	0
UT	0
GG	6
GF	2
GT	8

18.9.3   Rejected - #3954

Created - 14-09-2016, closed - 16-09-2017

Google employee and Polymer contributor justinfagnani created pull request #3954 where he added decorators that allow users of Polymer to define elements with TypeScript classes. This pull request does not fix existing issues but rather introduces new features, scheduled to be included in Polymer v2.0.

The code was reviewed extensively and approved by Google employee rictic. In March 2017, a few changes in TypeScript made a part of this pull request redundant. justinfagnani stated that the decorators added in this pull request did not have to change, but his plan was to make this pull request a separate repository, as they are not required to use in Polymer v2.0 and decorators are "non-standard". In September 2017, the separate repository polymer-decorators was introduced that includes the change from the main Polymer pull request. This pull request came from within Google and was not rejected, rather put in a separate repository for the interested users.

Label	#comments
AC	3
AT	0
AG	0
UF	4
UT	6
GG	1
GF	1
GT	2

18.9.4   Approved - #2689

Created - 05-11-2015, merged - 04-12-2015, first included in Polymer v1.2.4

Google Employee TimvdLippe created pull request #2689 where he fixed the incorrect parsing of minimized CSS output. It is a small change (26 lines of code have changed across three different files).

One day after the pull request was created, contributor nazar-pc made a pull request to Tim's pull request to improve the code quality slightly. After a small functional discussion between TimvdLippe and nazar-pc,

the pull request got merged by Google employee azakus.

The next day the merge of the pull request got reverted because it failed some continuous integration tests. Later a second pull request with this fix got merged (#3241).

Label	#comments
AC	1
AT	2
AG	5
UF	1
UT	3
GG	6
GF	0
GT	1

Chapter 19

PowerShell

By Sytze Andringa, Youri Arkesteijn, Bram Crielaard, and Luka Miljak
Delft University of Technology, 2019

Official PowerShell Core logo (on GitHub)

19.1   Table of Contents

19.2   Introduction
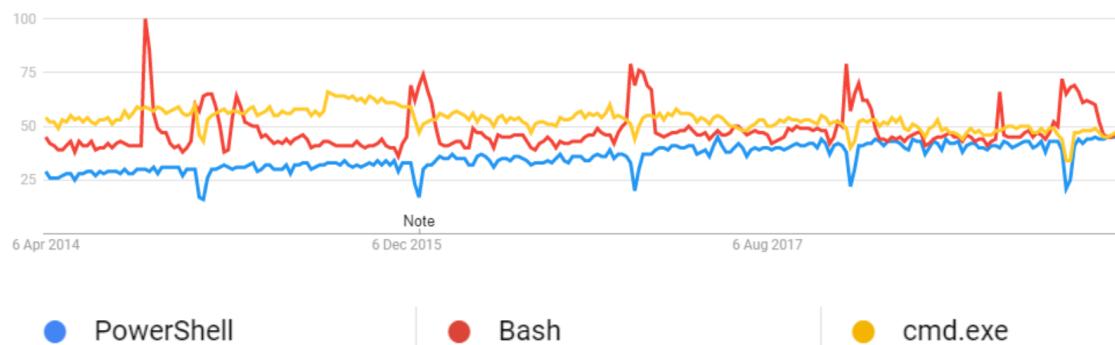
This chapter aims to describe and evaluate the architecture of PowerShell. To quote their own description:

PowerShell Core is a cross-platform (Windows, Linux, and macOS) automation and config-
uration tool/framework that works well with your existing tools and is optimized for dealing
with structured data (e.g. JSON, CSV, XML, etc.), REST APIs, and object models. It includes a
command-line shell, an associated scripting language and a framework for processing cmdlets.

The framework was initially released in 2006 for Windows[1]. A decade later, in 2016, PowerShell went
open-source and became available for Linux and macOS platforms as well. It now has the benefit of an
entire community contributing to the system. Figure 0.1 shows a simple overview of the history.

Figure 0.1 History of PowerShell[2].

Figure 0.2 Google Trends of PowerShell and two of its competitors: cmd.exe and Bash.

The Google Trends figure shows that PowerShell is slightly gaining in popularity. As with any large and
popular project, investigating and also evaluating its architecture is very interesting.

The first section of this chapter showcases all the related stakeholders of PowerShell. The second section
contains the context view, which describes the scope of the project and also its relation to external entities.
The development view can be found in the third section, which dissects the system into multiple modules,
describes some common design patterns in the code and also contains a short description of the build
practices. This chapter also analyzes two types of debt in PowerShell: The technical debt in section four and
the functional debt in section five. The final section gives a short conclusion on the architecture as a whole.

---

[1]Wikipedia, "PowerShell: Versions." [Online]. Available: https://en.wikipedia.org/wiki/PowerShell#Versions. [Accessed: 10-Apr-
2019]

[2]Microsoft, "Microsoft Technet Blogs." [Online]. Available: https://blogs.technet.microsoft.com/. [Accessed: 10-Apr-2019]

19.3 Stakeholder Analysis

19.3.1 Stakeholder Types

Rozanski and Woods[3] define numerous types of stakeholders. This section contains an analysis of the different types mentioned and how they apply to the PowerShell project.

19.3.1.1 Acquirers

The acquires of the system oversee the creation of PowerShell. This consists of both the PowerShell Committee and Microsoft at large. All changes go through the PowerShell Committee before they are included in the system.

19.3.1.2 Assessors

The assessors of the system make sure the product conforms to standards and legal regulation, which in the case of PowerShell is the Microsoft legal team.

19.3.1.3 Communicators

The communication about the system to other stakeholders is done by the PowerShell Committee and the PowerShell Maintainers. They provide information about any updates or decisions made to community as a whole. They also enforce that documentation is written when new functionality is added or undocumented behavior is found.

19.3.1.4 Developers

The developers of PowerShell consist of both Microsoft employees who focus on PowerShell as well as volunteers. Even though Microsoft employees make the final decision about the architecture of the system, a number of external contributors are allowed to decide when and if code should be merged. These contributors have been granted access by the committee to make these decisions.

19.3.1.5 Maintainers

The maintainers of the repository make sure the code evolves in a structured manner. In the case of PowerShell these maintainers consist of both the PowerShell committee (who make the final decision on whether or not design changes can occur) and the repository maintainers. The repository maintainers are (at the time of writing this report): @daxian-dbw, @TravisEz13, @adityapatwardhan, @iSazonov, and @anmenaga.

---

[3]N. Rozanski and E. Woods, *Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives*, 2nd ed. Addison-Wesley Professional, 2011.

19.3.1.6 Suppliers

PowerShells runs on different operating systems. The Get Powershell section in the README contains a full list of supported platforms. These platforms are considered suppliers, and because of limitations these platforms impose they should be considered as important stakeholders. Especially the Windows operating system is of interest, as PowerShell comes pre-installed on this platform.

19.3.1.7 Support Staff

Microsoft themselves handle the support for PowerShell. It is possible for end users to purchase a support license. Of course community support also exists by ways of Gitter, Slack, Reddit, GitHub, or fora such as Stack Overflow.

19.3.1.8 Testers

The testers make sure that PowerShell is fit for use. There are Microsoft employees who focus specifically on testing and continuous integration, such as @JamesWTruher. Tests need to be created or adapted with every pull requests that introduces new functionality, and developers are encouraged to write their own tests. There is a rigorous testing policy for contributions to PowerShell [4].

19.3.1.9 Users

There are many companies that use PowerShell in their working environment and as such they can be seen as major stakeholders. A good example is Michael Klement (@mklement0 on GitHub), who has worked with PowerShell in a few different external professional environments[5]. These users influence which functionality needs to be added to the language, as well as possible changes which need to be introduced. However, the PowerShell committee still has the final say over any decision made.

19.3.2 Types Not Mentioned by Rozanski and Woods.

19.3.2.1 Dependents

Dependents are users of PowerShell who do not necessarily know they are using PowerShell. The policy upheld by the PowerShell committee is that no backwards compatibility breaking changes can be introduced. This policy exists because dependents might use PowerShell scripts which they have not written themselves, nor know how to update. As such, they will still encounter problems if breaking changes are introduced during an update to PowerShell.

---

[4]Microsoft, "PowerShell testing guidelines." 2018 [Online]. Available: https://github.com/PowerShell/PowerShell/blob/master/docs/testing-guidelines/testing-guidelines.md

[5]M. Klement, "LinkedIn profile." 2019 [Online]. Available: https://www.linkedin.com/in/mklement0/

19.3.2.2 Development Tools

Tools which target PowerShell are –among others– ISESteroids[6], Visual Studio Code[7], and PowerShell Studio[8]. These systems provide tools for easy writing and editing PowerShell scripts.

19.3.2.3 Competitors

Competitors of PowerShell include shells such as Bash, Zsh, and Fish. They need to actively monitor each other's evolution to see if they need to adapt their own system to stay competitive.

19.3.3 Integrators For PowerShell

To decide which pull requests get merged the PowerShell project has two sets of members. Namely the PowerShell Committee and the Repository maintainers. If architectural changes are made to the system, or if new parameters are introduced to existing functions, they need to be approved by the PowerShell Committee. The committee maintains an overview of the project. They also make sure that all design changes fit the project and stay consistent with the rest of the codebase.[9]

For all other pull requests, the maintainers can take the decision to merge. The maintainers make sure the code style is consistent and the code is of high quality. This means they have to make sure details such as variable names, design patterns, and function names are consistent when introducing new code.[10]

When analyzing the pull requests created for PowerShell we concluded that this is not just a guideline, but is actually used in practice.

19.3.4 Relevant People We Would Like To Contact

We have concluded there are three people who we believe are interesting to contact. These people are @iSazonov, @mklement0, and @SteveL-MSFT.

iSazonov is currently a maintainer of the PowerShell project on GitHub, however he is not an employee of Microsoft. For every pull request we have read it was difficult to find one in which he had not participated.

mklement0 is extremely active in the community and has been using PowerShell in a professional setting since 2010. Hearing from someone who has actively seen the system evolve as both a user as well as a developer would result in an interesting insight.

SteveL-MSFT is a member of the PowerShell Committee and actively participates in discussion on most pull requests. Hearing the input of one of the core team members would shine light on the decisions made when designing the PowerShell architecture.

---

[6]PowerTheShell, "ISESteroids." [Online]. Available: http://www.powertheshell.com/isesteroids/. [Accessed: 28-Mar-2019]

[7]Microsoft, "Visual Studio Code." [Online]. Available: https://code.visualstudio.com/. [Accessed: 28-Mar-2019]

[8]Sapien Technologies, Inc., "PowerShell Studio 2019." [Online]. Available: https://www.sapien.com/software/powershell_studio. [Accessed: 28-Mar-2019]

[9]Microsoft, "PowerShell Governance." [Online]. Available: https://github.com/PowerShell/PowerShell/blob/master/docs/community/governance.md#powershell-committee. [Accessed: 28-Mar-2019]

[10]Microsoft, "Repository Maintainers." [Online]. Available: https://github.com/PowerShell/PowerShell/blob/master/docs/maintainers/README.md. [Accessed: 28-Mar-2019]

19.3.5 Pull Request Analysis

We analyzed 10 open and 10 closed pull requests. From analyzing these we conclude the following:

1. The main focus is that pull request actually solve an issue (an issue should be referenced from a pull request).[11]–[12]
2. All changes should be tested (to ensure a high coverage and minimize the amount of unexpected behavior).[13], [14], [15]–[16], [17]–[18],ˆ[4]–[19]
3. There should not be backwards compatibility changes, PowerShell **must** be backwards compatible.[20], [21], [22], [23]
4. Comments should be correct, or removed if not needed.[24]–[25], [26], [27]–[28]
5. Code should be consistent, the naming of parameters and default values of parameters give way to

[11]I. Sazonov, "PowerShell pull request #3690." [Online]. Available: https://github.com/PowerShell/PowerShell/pull/3690. [Accessed: 27-Mar-2019]

[12]S. Barizien, "PowerShell pull request #8745." [Online]. Available: https://github.com/PowerShell/PowerShell/pull/8745. [Accessed: 27-Mar-2019]

[13]I. Sazonov, "PowerShell pull request #3690." [Online]. Available: https://github.com/PowerShell/PowerShell/pull/3690. [Accessed: 27-Mar-2019]

[14]C. Bergmeister, "PowerShell pull request #4612." [Online]. Available: https://github.com/PowerShell/PowerShell/pull/4612. [Accessed: 27-Mar-2019]

[15]M. Klement, "PowerShell pull request #4761." [Online]. Available: https://github.com/PowerShell/PowerShell/pull/4761. [Accessed: 27-Mar-2019]

[16]A. patwardhan, "PowerShell pull request #5760." [Online]. Available: https://github.com/PowerShell/PowerShell/pull/5760. [Accessed: 27-Mar-2019]

[17]I. Sazonov, "PowerShell pull request #7702." [Online]. Available: https://github.com/PowerShell/PowerShell/pull/7702. [Accessed: 27-Mar-2019]

[18]A. Gauthier, "PowerShell pull request #8199." [Online]. Available: https://github.com/PowerShell/PowerShell/pull/8199. [Accessed: 27-Mar-2019]

[19]J. Swallow, "PowerShell pull request #7509." [Online]. Available: https://github.com/PowerShell/PowerShell/pull/7509. [Accessed: 27-Mar-2019]

[20]R. Holt, "PowerShell pull request #8142." [Online]. Available: https://github.com/PowerShell/PowerShell/pull/8142. [Accessed: 27-Mar-2019]

[21]A. Gauthier, "PowerShell pull request #8199." [Online]. Available: https://github.com/PowerShell/PowerShell/pull/8199. [Accessed: 27-Mar-2019]

[22]J. Swallow, "PowerShell pull request #7509." [Online]. Available: https://github.com/PowerShell/PowerShell/pull/7509. [Accessed: 27-Mar-2019]

[23]D. Stenberg, "PowerShell pull request #1901." [Online]. Available: https://github.com/PowerShell/PowerShell/pull/1901. [Accessed: 27-Mar-2019]

[24]C. Bergmeister, "PowerShell pull request #5051." [Online]. Available: https://github.com/PowerShell/PowerShell/pull/5051. [Accessed: 27-Mar-2019]

[25]A. patwardhan, "PowerShell pull request #5760." [Online]. Available: https://github.com/PowerShell/PowerShell/pull/5760. [Accessed: 27-Mar-2019]

[26]I. Sazonov, "PowerShell pull request #7702." [Online]. Available: https://github.com/PowerShell/PowerShell/pull/7702. [Accessed: 27-Mar-2019]

[27]A. Gauthier, "PowerShell pull request #8199." [Online]. Available: https://github.com/PowerShell/PowerShell/pull/8199. [Accessed: 27-Mar-2019]

[28]J. Swallow, "PowerShell pull request #7509." [Online]. Available: https://github.com/PowerShell/PowerShell/pull/7509. [Accessed: 27-Mar-2019]

many discussions.[29]–[30], [31], [32]–[33], [34], [35]

6. If a pull request becomes too complex, it is closed and split up into multiple pull requests.[36], [37], [38], [39], [40]

7. Design changes and improvements can be discussed through the RFC (Request For Comments) process, which provides the community with a way to leave feedback on documents instead of written code.[41], [42], [43]

8. The PowerShell team is open to outsiders challenging their calls. Even if you have never contributed any code, if you disagree with their decisions and can explain why, they will reconsider.[44], [45], [46], [47]

9. People who have opened the issue are contacted to see if an issue is actually resolved by a pull request, and not just believed to be solved by the author.[48]

10. Community members are contacted by the core team for their input, not just the seniors make every decision without talking to the rest of the community.[49]

---

[29]C. Bergmeister, "PowerShell pull request #5051." [Online]. Available: https://github.com/PowerShell/PowerShell/pull/5051. [Accessed: 27-Mar-2019]

[30]A. patwardhan, "PowerShell pull request #5760." [Online]. Available: https://github.com/PowerShell/PowerShell/pull/5760. [Accessed: 27-Mar-2019]

[31]I. Sazonov, "PowerShell pull request #7702." [Online]. Available: https://github.com/PowerShell/PowerShell/pull/7702. [Accessed: 27-Mar-2019]

[32]A. Gauthier, "PowerShell pull request #8199." [Online]. Available: https://github.com/PowerShell/PowerShell/pull/8199. [Accessed: 27-Mar-2019]

[33]R. Holt, "PowerShell pull request #3169." [Online]. Available: https://github.com/PowerShell/PowerShell/pull/3169. [Accessed: 27-Mar-2019]

[34]J. Swallow, "PowerShell pull request #7509." [Online]. Available: https://github.com/PowerShell/PowerShell/pull/7509. [Accessed: 27-Mar-2019]

[35]J. L. Whitlock, "PowerShell pull request #3125." [Online]. Available: https://github.com/PowerShell/PowerShell/pull/3125. [Accessed: 27-Mar-2019]

[36]M. Klement, "PowerShell pull request #4761." [Online]. Available: https://github.com/PowerShell/PowerShell/pull/4761. [Accessed: 27-Mar-2019]

[37]R. Holt, "PowerShell pull request #8142." [Online]. Available: https://github.com/PowerShell/PowerShell/pull/8142. [Accessed: 27-Mar-2019]

[38]R. Holt, "PowerShell pull request #3169." [Online]. Available: https://github.com/PowerShell/PowerShell/pull/3169. [Accessed: 27-Mar-2019]

[39]J. Swallow, "PowerShell pull request #7509." [Online]. Available: https://github.com/PowerShell/PowerShell/pull/7509. [Accessed: 27-Mar-2019]

[40]R. Dunham, "PowerShell pull request #8886." [Online]. Available: https://github.com/PowerShell/PowerShell/pull/8886. [Accessed: 27-Mar-2019]

[41]R. Holt, "PowerShell pull request #3169." [Online]. Available: https://github.com/PowerShell/PowerShell/pull/3169. [Accessed: 27-Mar-2019]

[42]D. Stenberg, "PowerShell pull request #1901." [Online]. Available: https://github.com/PowerShell/PowerShell/pull/1901. [Accessed: 27-Mar-2019]

[43]J. L. Whitlock, "PowerShell pull request #3125." [Online]. Available: https://github.com/PowerShell/PowerShell/pull/3125. [Accessed: 27-Mar-2019]

[44]GitHub user @zhenggu, "PowerShell pull request #5525." [Online]. Available: https://github.com/PowerShell/PowerShell/pull/5525. [Accessed: 27-Mar-2019]

[45]A. patwardhan, "PowerShell pull request #5760." [Online]. Available: https://github.com/PowerShell/PowerShell/pull/5760. [Accessed: 27-Mar-2019]

[46]P. Braathen, "PowerShell pull request #8131." [Online]. Available: https://github.com/PowerShell/PowerShell/pull/8131. [Accessed: 27-Mar-2019]

[47]A. Gauthier, "PowerShell pull request #8199." [Online]. Available: https://github.com/PowerShell/PowerShell/pull/8199. [Accessed: 27-Mar-2019]

[48]I. Sazonov, "PowerShell pull request #7702." [Online]. Available: https://github.com/PowerShell/PowerShell/pull/7702. [Accessed: 27-Mar-2019]

[49]P. Braathen, "PowerShell pull request #8131." [Online]. Available: https://github.com/PowerShell/PowerShell/pull/8131. [Accessed: 27-Mar-2019]

19.3.6   Power/Interest Grid

Figure 1.1 shows how the stakeholders mentioned in section 1.2 and 1.3 can be classified in terms of power and interest.
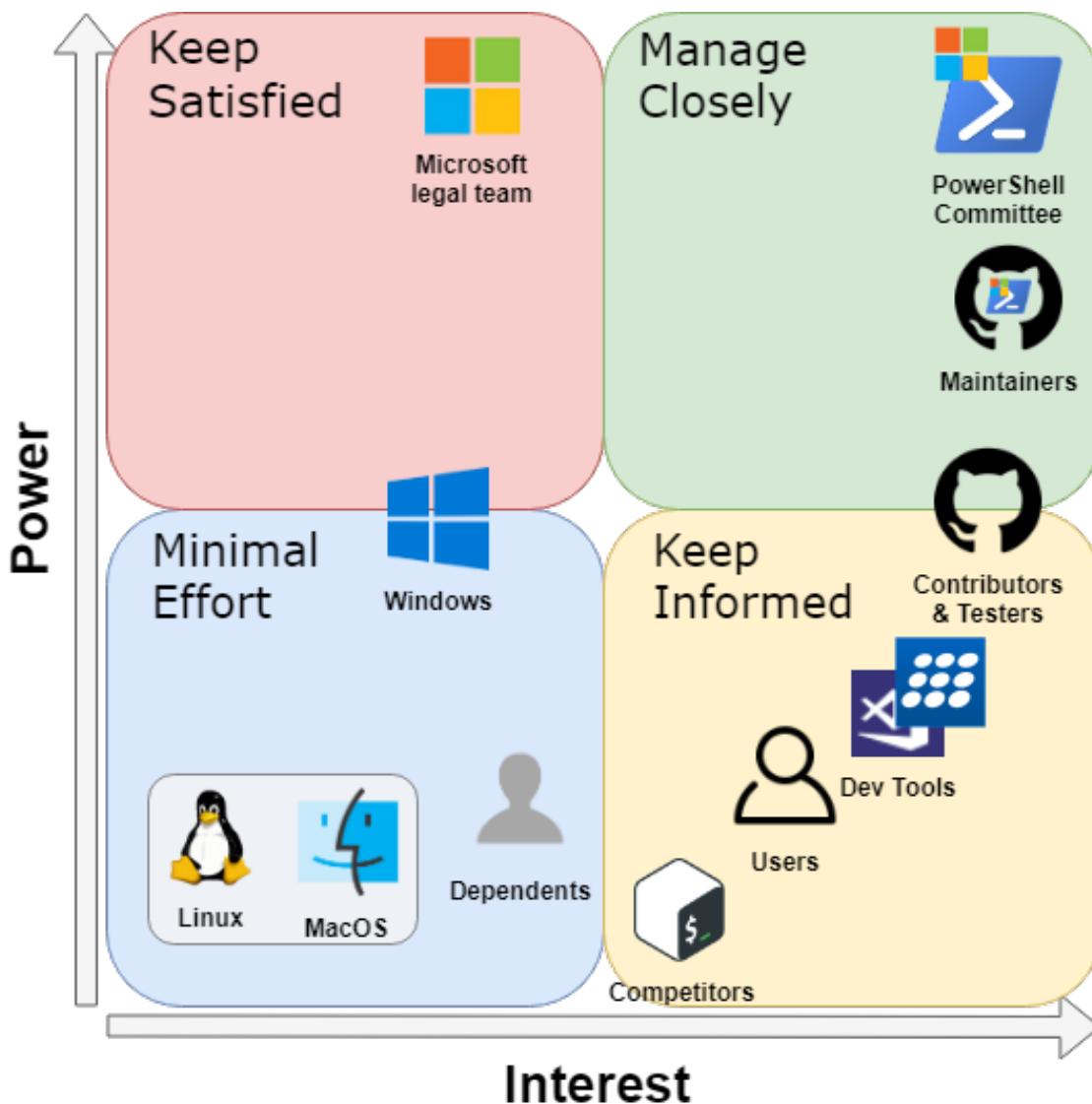
Figure 1.1. A power/interest grid of several entities that have a relation with the system.

19.4   Context View

This section describes the Context view of the system. The view defines what PowerShell does, which is described in section 2.1 using a scope definition. Furthermore, the view also describes the relation between

the system and external entities. These relations can be found in Section 2.2, where are Context diagram was used as the method of modeling these relations.

19.4.1   System Scope

The scope definition of PowerShell is described as a list of **key** requirements of the systems. These requirements are as follows:

- Provide a task automation and configuration management framework.[50]
- Provide availability on multiple platforms (Windows, Linux, and macOS).[51]
- Include a command-line shell.[52]
- Allow users to easily extend PowerShell with their own commands and frameworks.[53]
- Work with structured data (as opposed to plain text, as used by other traditional Shells).[54]
- Include a scripting language.[55]
- It should be easy to learn by users even when coming from a similar tool (like Bash).[56]
- Updates should be backward compatible.[57]

19.4.2   Context Diagram

The Context diagram (Figure 2.1) showcases the most important external entities of PowerShell and indicates what the relation between them and the system is.

---

[50]Microsoft, "PowerShell README." [Online]. Available: https://github.com/PowerShell/PowerShell/blob/master/README.md#-powershell. [Accessed: 08-Apr-2019]

[51]Microsoft, "PowerShell README." [Online]. Available: https://github.com/PowerShell/PowerShell/blob/master/README.md#-powershell. [Accessed: 08-Apr-2019]

[52]Microsoft, "PowerShell README." [Online]. Available: https://github.com/PowerShell/PowerShell/blob/master/README.md#-powershell. [Accessed: 08-Apr-2019]
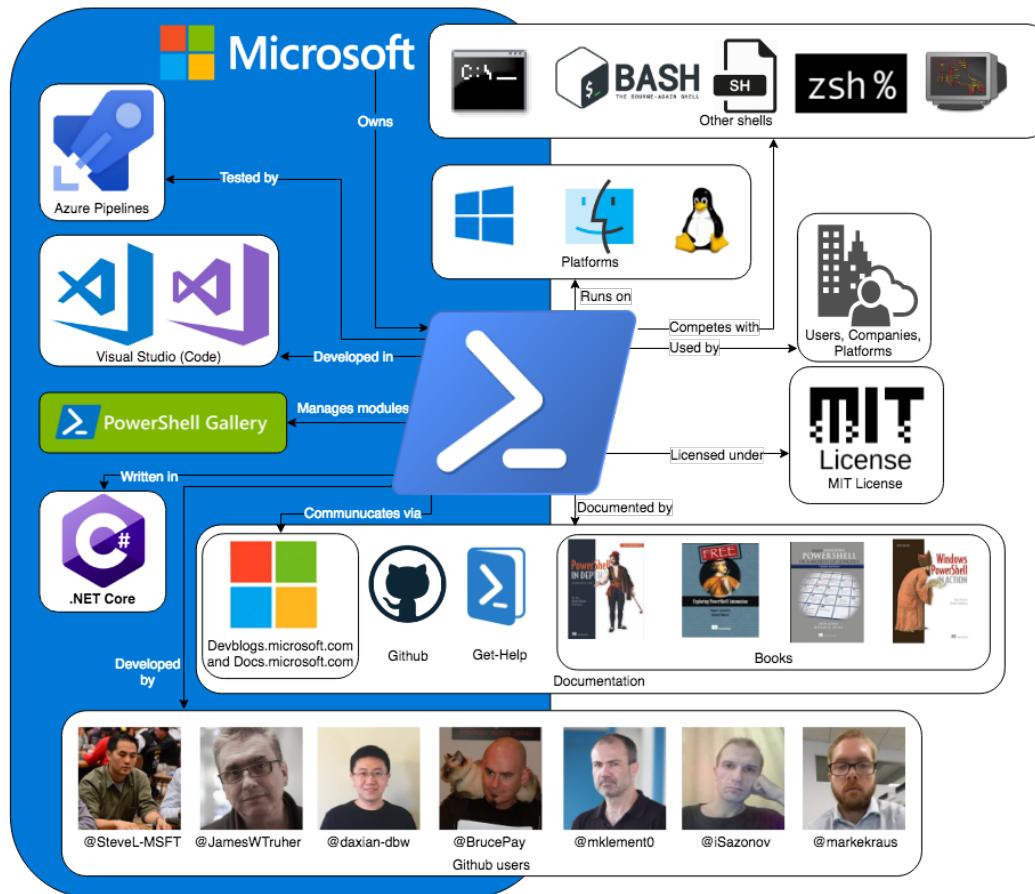
[53]Microsoft, "Cmdlet Example." [Online]. Available: https://github.com/PowerShell/PowerShell/blob/master/docs/cmdlet-example/command-line-simple-example.md. [Accessed: 08-Apr-2019]

[54]Microsoft, "PowerShell README." [Online]. Available: https://github.com/PowerShell/PowerShell/blob/master/README.md#-powershell. [Accessed: 08-Apr-2019]

[55]Microsoft, "PowerShell README." [Online]. Available: https://github.com/PowerShell/PowerShell/blob/master/README.md#-powershell. [Accessed: 08-Apr-2019]

[56]Microsoft, "Map Book for Experienced Bash users." [Online]. Available: https://github.com/PowerShell/PowerShell/tree/master/docs/learning-powershell#map-book-for-experienced-bash-users. [Accessed: 08-Apr-2019]

[57]Microsoft, "Breaking Change Contract." [Online]. Available: https://github.com/PowerShell/PowerShell/blob/master/docs/dev-process/breaking-change-contract.md. [Accessed: 08-Apr-2019]

Figure 2.1. A Context diagram for PowerShell. It consists of mostly of the stakeholders mentioned in chapter 1, but also the development language `C#`, Azure continuous integration[58], and several other resources (books, package manager, . . .) for PowerShell users[59]. PowerShell is licensed under the MIT license[60]. All entities inside the blue area are owned by Microsoft.

19.5   Development View

This section describes the Development Viewpoint of the system. Section 3.1 dissects PowerShell into multiple modules and shows the dependencies between them. 3.2 describes some design choices made for

---

[58]Microsoft, "Testing Guidelines."   [Online].   Available: https://github.com/PowerShell/PowerShell/blob/master/docs/testing-guidelines/testing-guidelines.md#ci-system. [Accessed: 08-Apr-2019]
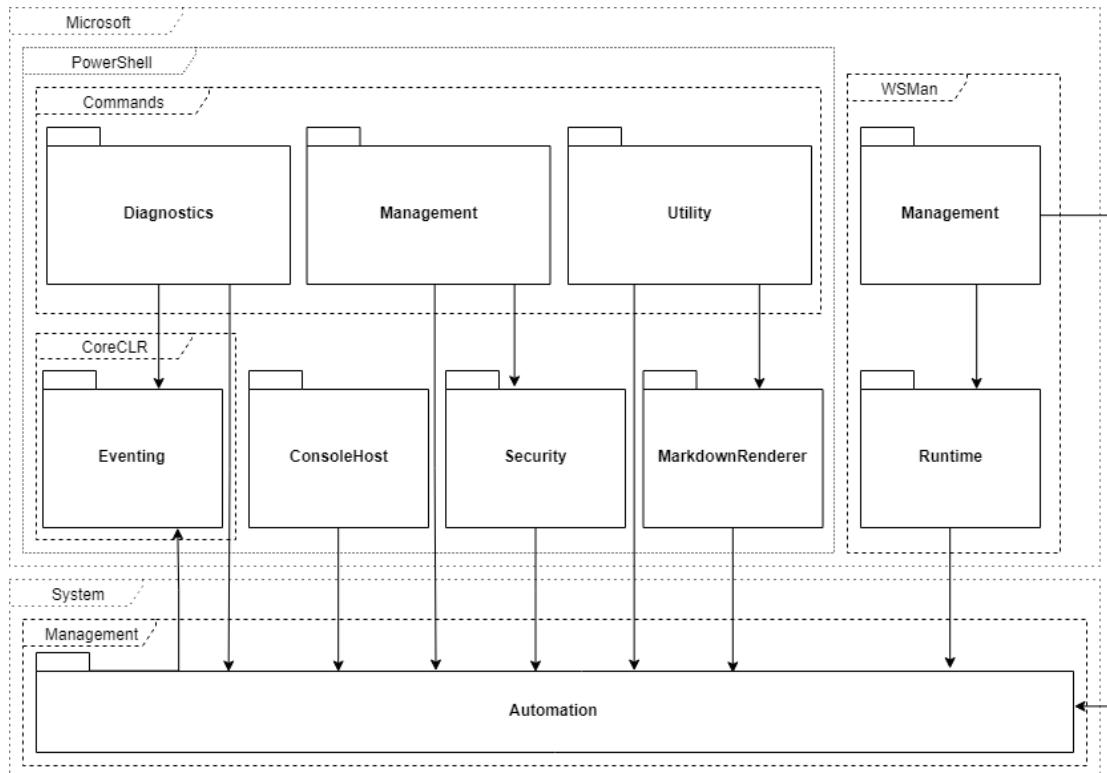
[59]Janik von Rotz, "Awesome PowerShell." [Online]. Available: https://github.com/janikvonrotz/awesome-powershell. [Accessed: 08-Apr-2019]

[60]Microsoft, "MIT license." [Online]. Available: https://github.com/PowerShell/PowerShell/blob/master/LICENSE.txt. [Accessed: 08-Apr-2019]

repeating problems. A short description of the build process is found in 3.3.

19.5.1 Module Structure

This section provides an overview of the module structure of PowerShell and what the dependencies between these modules are.

Figure 3.1. A Module Structure representing PowerShell.

As visible in the figure, we view each namespace within PowerShell as a separate module. It makes sense to use this C# language construct in order to organize classes. The largest namespace is `Microsoft`, which in turn contains two namespaces `PowerShell` and `WSMan` (Web Services Management). `WSMan` provides utility for remote management for both the host as well as the client side.

The `PowerShell` namespace in turn contains several other namespaces. `ConsoleHost` provides an access point from the console to the PowerShell engine process, as referenced here. `Eventing` allows for asynchronous events to be created and handled. The `Commands`namespace contains *most* of the built in cmdlets of PowerShell. PowerShell distinguishes three types of commands where all three are represented using namespaces. `Diagnostics` contains commands for managing data in event logs. `Management` provides commands for managing Windows settings such as `Move-Item`, `New-Service`, and `Rename-Computer`. The last command type is `Utility`, which provides commands for managing the basic features of PowerShell such as `Set-Alias`, `ConvertTo-Json` and unexpectedly `Set-Date`.

There are two more modules outside of the commands namespace, namely `Security` and

MarkdownRenderer.   Security provides commands that manage the basic security features in Windows, like execution policies and signatures. MarkdownRenderer provides commands used to transform from Markdown to PowerShell's internal notation. This post describes how to these features in detail.

Lastly there is System.Management.Automation namespace. Automation is the place where commands are evaluated, errors are created, and results are produced. Every other module in the system depends on this module, which clearly makes it the bottom layer of PowerShell.

There are some unexpected architectural design choices made. Notable is the placement of packages containing commands outside of the command package and the number of namespaces named Management. Also, the Automation package has very high coupling with the Microsoft.PowerShell namespace, but is not contained inside it. Furthermore, there is an unclear namespace for eventing.

Lastly, the folder structure of the project does not correspond one-to-one with the namespaces. Within namespaces there exist folders that contain classes that still belong to that same namespace. Unlike for instance Java, C# permits this type of file organization. In PowerShell, folders are used to group classes so that they can be found more easily. However, these classes still solely exist to serve the namespace. Similarly to this, .cs files also often contain more than one class. We would argue that there are some cases where a separate namespace is preferred over just a folder, such as the engine folder inside of the Automation namespace, but in the end it all boils down to preferences.

This analysis of the modules within PowerShell and the dependencies between them was done using code maps in Visual Studio Enterprise. The actual description of each individual module was done by manually reviewing the code, as there is no larger document on the PowerShell repository mentioning the namespace overview or the folder structure.

19.5.2   Common Design

This section identifies several commonalities in the code across different implementations. While PowerShell only imposes some minor design constraints in their coding guidelines, looking at the code one finds that there are some choices made that reflect some designs that the PowerShell developers prefer.

Defining cmdlets. Cmdlets in PowerShell (both internal and external) are defined by creating a class that extends the Cmdlet class contained within the Automation namespace. These classes are then processed using reflection in C#. Commands are the only part of the code that intensively use reflection and C# attributes (which are similar to Java annotations). This document contains an example on how users can define their own cmdlet. Built-in PowerShell commands use the exact same pattern of defining them.

Input validation. The input PowerShell gets comes in the form of commands. This is why the input validation process happens in the class that defines that particular cmdlet. Developers can put constraints on arguments given to parameters by adding C# attributes to them. For example, the ConvertTo-Json command takes a Depth parameter. The code snippet below shows how this parameter is defined in the ConvertToJsonCommand class contained in the Commands.Utility namespace.

Notable is the usage of the ValidateRange(1, int.MaxValue) attribute, which is used to validate that the given depth is at least one. In the Automation namespace, the Attributes.cs file defines many of these attributes that are commonly used for input validation. Obviously there are cases that are not that common. For these, traditional if statements are used and exceptions thrown if the input is in an incorrect format.

```
[Parameter]
[ValidateRange(1, int.MaxValue)]
public int Depth
{
    get { return _depth; }
    set { _depth = value; }
}
```

Figure 19.1: Code snippet

Platform Dependent code. It often arises that there needs to a different flow depending on the operating system. PowerShell chooses to use preprocessor directives to get platform-specific builds. The creates code constructs of the following form:

```
#if UNIX
     ..... // UNIX specific code here
#else
     .....
```

However, the code guidelines mention that runtime checks are acceptable if it greatly improves the readability without causing performance concerns in performance-sensitive code.

Dependency Inversion. The DI principle states that classes should depend on abstractions rather than concrete classes. PowerShell only follows this partially. From analyzing the code, it is clear that interfaces are only used if there are two or more implementations of the interface.

Interface Extension. Sometimes interfaces need to be extended with new behavior. An example is the `ICommandRuntime` interface in the `Automation` namespace, which defines the set of functionality that must be implemented to directly execute an instance of `Cmdlet`. Later in the development process, the need arose to add functionality to this interface to add support writing an informational record to the host or user. Rather than adding this to the interface itself, PowerShell chooses to extends the `ICommandRuntime` interface with a new interface, appropriately named `ICommandRuntime2`. Note that this mostly only applies to interfaces and not classes.

19.5.3 Codeline Overview

PowerShell has build instructions for each mayor operating system[61]. Using Azure Pipelines, they run nightly builds for each operating system[62]. As per their Testing Guidelines, contributors are expected to write automated tests for their code, which are actively checked in Pull Requests.

---

[61]Microsoft, "Build Instructions." [Online]. Available: https://github.com/PowerShell/PowerShell#building-the-repository. [Accessed: 10-Apr-2019]

[62]Microsoft, "Build Status." [Online]. Available: https://github.com/PowerShell/PowerShell#build-status-of-nightly-builds. [Accessed: 10-Apr-2019]

19.6   Technical Debt

Technical debt builds when development teams decide to implement an easy solution to their problem without thinking about extendability, either because of time constraints or inexperience. These "easy" solutions can take the shape of code which can not be extended without major refactors, or entire projects which are no longer worthwhile to maintain.

19.6.1   Code Quality Debt

The repository was analyzed using CodeFactor and SonarQube[63]. The main issues found took the form of overly complex methods and style errors. Both of these hinder the extensibility and readability of the code. Ignoring auto-generated files, there are methods with a cyclomatic complexity up to 405, and classes with a complexity up to 1743. There also are many exceptions which are ignored without explanation, making it difficult to understand the intentional behavior.

An example of a specific file which contains multiple critical errors can be found here. Among other issues it contains getters which do not return the expected value (either different fields or null), as well as setters which set unrelated fields to their name. This is bug prone and should be refactored.

During a manual analysis multiple violations of the Liskov substitution principle were found. For example, in the file `ProviderBaseSecurity.cs` on line 40 the method assumes that a subclass of Cmdlet implements the ISecurityDescriptorCmdletProvider, which violates this principle. A similar violation can be found in `ProviderBase.cs` on line 262.

There is also debt which has been confirmed, such is in `RemoteSessionCapability.cs`. As old releases can not be changed, the debt addressed in this comment can not be resolved in later releases.

```
internal RemoteSessionCapability(RemotingDestination remotingDestination)
{
    _protocolVersion = RemotingConstants.ProtocolVersion;
    // PS Version 3 is fully backward compatible with Version 2
    // In the remoting protocol sense, nothing is changing between PS3 and PS2
    // For negotiation to succeed with old client/servers we have to use 2.
    _psversion = new Version(2,0); // PSVersionInfo.PSVersion;
```

Remove this commented out code. ···   last year ▾ L62 ⚲
⊕ Code Smell ▾ ⊘ Major ▾ ○ Open ▾ Not assigned ▾ 5min effort Comment   🏷 misra, unused ▾

```
    _serversion = PSVersionInfo.SerializationVersion;
    _remotingDestination = remotingDestination;
}
```

Figure 4.1. Hard-coded version numbers.

By analyzing the historic data provided by CodeFactor we can see the number of issues has decreased to 50% of its maximum, but is still not up to the defined standards. The initial spike in issues can be traced

---

[63]SonarQube, "PowerShell SonarQube report." [Online]. Available: https://sonarcloud.io/dashboard?id=Geweldig_PowerShell. [Accessed: 17-Mar-2019]

back to this commit, where more rules were introduced.

Figure 4.2. Code issue history.

19.6.2   Testing Debt

Because of the high complexity of the codebase it is infeasible to write exhaustive tests. For example this file contains a method with a complexity of 405, meaning there should be 2405 (~8x10121) test cases to test *this single method* exhaustively. Refactoring this method would improve testability and readability, as it has already been informally split by using comments.
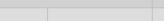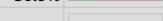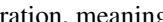
The file `CodeCoverageAnalysis.md` reports *"hot spots of missing coverage"*. It also states that *"The metrics used for selection of these hot spots were: # missing lines and likelihood of code path usage."*. This is risky, as bugs on infrequently used paths can lead to major issues further along in the program.

The reported line coverage is 63.2%, which is much lower than one would hope. For example, when opening this pull request breaking changes were almost introduced which were not covered by tests, and went unnoticed by multiple reviewers before being resolved. Interestingly, the reported coverage was not updated for over four months and has been manually updated for the release of PowerShell 6.2.0. It also does not take into account branch coverage but only looks at line coverage.

Files	≡	●	●	●	Coverage
📁 Microsoft.Management.Infrastructure.CimCmdlets	5,298	2,553	0	2,745	48.18%
📁 Microsoft.PowerShell.Commands.Diagnostics	1,258	668	0	590	53.10%
📁 Microsoft.PowerShell.Commands.Management	12,214	7,457	0	4,757	61.05%
📁 Microsoft.PowerShell.Commands.Utility	18,078	12,836	0	5,242	71.00%
📁 Microsoft.PowerShell.ConsoleHost	6,758	3,288	0	3,470	48.65%
📁 Microsoft.PowerShell.CoreCLR.Eventing	3,123	1,290	0	1,833	41.30%
📁 Microsoft.PowerShell.MarkdownRender	348	246	0	102	70.68%
📁 Microsoft.PowerShell.Security	3,322	1,710	0	1,612	51.47%
📁 Microsoft.WSMan.Management	6,419	3,993	0	2,426	62.20%
📁 System.Management.Automation	229,4...	146,8...	0	82,555	64.01%
📄 Microsoft.WSMan.Runtime/WSManSessionOption.cs	41	41	0	0	100.00%
📄 powershell/Program.cs	3	3	0	0	100.00%
Folder Totals (12 files)	286,2...	180,9...	0	105,3...	63.20%
Project Totals (996 files)	286,2...	180,9...	0	105,3...	63.20%

Figure 4.3. Coverage generated by Codecov.

Furthermore, when manually running the coverage tools according to the documentation provided, we arrive at a line coverage of 37.7% and a branch coverage of 29.7%. This shows a discrepancy in the tools used.

Name	Covered	Uncovered	Coverable	Total	Line coverage	Branch coverage
+ Microsoft.Management.Infrastructure.CimCmdlets	2692	3525	6217	57701	43.3%	25.5%
+ Microsoft.PowerShell.Commands.Diagnostics	249	1093	1342	2909	18.5%	16.1%
+ Microsoft.PowerShell.Commands.Management	4862	9407	14269	261855	34%	26.6%
+ Microsoft.PowerShell.Commands.Utility	3238	16421	19659	117226	16.4%	14.5%
+ Microsoft.PowerShell.ConsoleHost	1709	5701	7410	23977	23%	17.1%
+ Microsoft.PowerShell.CoreCLR.Eventing	931	2245	3176	8068	29.3%	20.9%
+ Microsoft.PowerShell.MarkdownRender	0	348	348	1619	0%	0%
+ Microsoft.PowerShell.Security	492	3358	3850	62792	12.7%	8.8%
+ Microsoft.WSMan.Management	2095	4522	6617	103231	31.6%	30.9%
+ Microsoft.WSMan.Runtime	0	41	41	226	0%	
+ System.Management.Automation	104763	153258	258021	3701717	40.6%	31.9%
+ pwsh	3	0	3	25	100%	

Figure 4.4. Coverage generated manually.

Historically the coverage jumps between 40% and 60%. There are single commits which make the line coverage either drop or raise 20%, which shows there are issues with the way the coverage is calculated.It is also worth nothing that this file states a subset of the test suite is run for continuous integration, meaning functionality can break without being noticed by the CI.

Bump Newtonsoft.Json from 11.0.2 to 12.0.1 (#8348)
dependabot[bot] 4 months ago   master   de5f69c   ✔ CI Passed   62.51%   ∅   +17.96%

Add SkipCA and SkipCN check requirement to WinRM/OMI HTTPS connection (#8279)
PaulHigin 4 months ago   master   0657163   ✔ CI Passed   44.55%   ∅   +0.08%

Figure 4.5. The coverage jumping 18% between commits.

19.6.3   Usage of Tools by The Community

As is best practice PowerShell makes use of automated tools which aid the developer in writing code according to the standards defined by the development team. However, the development team does not take these rules to heart and disagrees with the results. There are pull requests to tweak these tools and better align them with the views of the development team. Unfortunately, these pull requests are often left open for months as no decision can be reached.

As a result of the development team not agreeing with their tools there are many (easy to resolve) open issues, which are not prioritized. Furthermore, the data these tools provide might even be completely incorrect. As previously mentioned the test coverage had not been updated for four months. Because of this the metrics do not represent the state of the codebase, meaning they are untrustworthy. These tools should lead to a cleaner and better codebase, but are ignored and deemed inconvenient.

A major issue which arose because of the decision to ignore the metrics is that old code suffers from technical debt which can not be resolved. For example, hard-coded version numbers can not be resolved, as PowerShell prioritizes backwards compatibility which would be broken by resolving the issue. This means if the analysis tools continue to be ignored, the codebase will continue to get worse over time with no way to fix it. As such, the tools should be configured correctly and their results should be used as suggestions for improvements, instead of being treated as annoyances.

19.6.4   Recommendations

The development team should prioritize paying their technical debt. The issues raised by CodeFactor should either be resolved, or rules should be adapted to accurately reflect the intentions of the development team. Files and methods should be split up to make the flow of code through the system easier to understand and to reduce the cyclomatic complexity. Also, the test coverage should be accurately reported, as well as more tests added, as there is uncommon intended behavior which is not covered by tests. All of these changes would result in a more newcomer friendly project, while also making it easier to spot mistakes early on in the process.

19.7   Functional Debt

Functional debt is not a widely used term in software development. Roman Predein mentioned that there are only a few references to this phenomenon and that they do not match how he sees it. We therefore conclude that, at the moment, functional debt has no "real" definition. Therefore, we explain in 5.1 what we regard as functional debt. 5.2 explains the relation between this debt and backwards compatibility. The rest of the subsections explain how it relates to PowerShell.

19.7.1   Defining Functional Debt

We consider functional debt to be *the amount of mismatches between expected and actual functionality*.

By *expected functionality* we mean what functionality the user would expect. The *actual functionality* is the functionality of the system. To explain the concept, we will distinguish three types of functional debt by the way they are introduced into the system.

1. *When the user expects to find a particular feature, but it does not exist.* This is a result of improperly documenting what the system should and should not be able to do.
2. *When the developer and the user have different expectations of the behavior for a particular functionality.* This can be caused by the user misinterpreting the documentation due to inaccuracies or vagueness. Inappropriate naming of functions also causes this difference.
3. *When the intended functionality of a function differs from the actual functionality.* This happens through mistakes and can be tackled by having low technical debt, as it then becomes harder for bugs to slip in.

As shown, usually functional debt can be prevented by having proper documentation and low technical debt. However, it is impossible to protect a project from every form of functional debt. A system should therefore acknowledge that functional debt can be introduced into the system, and should come up with a strategy to tackle it.

19.7.2   Backwards Compatibility

Functional debt can become an even bigger issue when considering backwards compatibility. If following backwards compatibility in the most strict sense possible, it would not be possible to fix any functional debt introduced earlier from the second or third categories as discussed earlier, since that would change the functionality of some already existing feature. If being less strict, it is possible to fix functional debt by allowing some forms of breaking changes. A policy for solving technical debt can be considered an important architectural design choice for systems where backwards compatibility plays a role.

To help us draw a relation between functional debt and backwards compatibility, we introduce a term called the *backwards compatibility cost*. This cost is a measurement of the amount of damage when the contribution is accepted in the context of backwards compatibility. In general, a contribution with high functionality changes corresponds to a high backwards compatibility cost and vice-versa.

To any contribution, we can assign a backwards compatibility cost and an amount of functional debt it solves. If we neglect technical debt, and if the backwards compatibility cost is lower than the amount of functional debt it would solve, we should accept it. Else, if higher, reject.

Figure 5.1. Strong backwards compatibility commitment makes it harder to fix functional bugs

19.7.3   PowerShell's Breaking Changes Contract

PowerShell's breaking changes contract explains how PowerShell handles backwards compatibility. We will view this contract from a functional debt perspective, which follows the rules defined earlier.

PowerShell has a serious commitment towards backwards compatibility. Any stable existing feature shipped has to be introduced in any next version of PowerShell. New features which are marked as *"in preview"* are still allowed to be modified. Breaking changes are classified into four *"buckets"*: Public Contract, Reasonable Grey Area, Unlikely Grey Area, Clearly Non-Public.

The Public Contract and Clearly Non-Public buckets follow a set of rules to decide whether it should be accepted. This can be interpreted as assigning either zero (when the breaking change is allowed) or infinity (when not allowed) backwards compatibility cost to contributions within these buckets.

For the two grey areas, the backwards compatibility cost has to be determined by judgement, and functional debt has to be weighed against the backwards compatibility cost.

As explained earlier, when dealing with backwards compatibility we should think about how to handle bugs. PowerShell allows this under some circumstances, by allowing the change of *"any existing behavior that results in an error message"*. On the other hand, *"It doesn't matter if the existing behavior is 'wrong' "* and *"we must be extremely sensitive to breaking existing users and scripts"*. This shows that PowerShell has thought of this, by providing some kind of upper and lower bound of when the fix is allowed.

19.7.4   Example

Pull request #1901 is a nice example where commitment towards backwards compatibility shows how functional debt grows over time.

@bagder thinks adding `curl` and `wget` was a *"breaking change"* and should therefore be removed. If we would map @bagder's view to ours, the expected functionality is that PowerShell should never overwrite commonly used commands. He received a lot of support from others and thus it is reasonable to think the contribution would reduce functional debt.

@lzybkr says that at the time of adding the alias, *"the design decision was consistent"*. At the time, @lzybkr had a different interpretation of expected functionality. However, he also says that it may have been misguided to add the aliases in the first place. Due to it being a breaking change, the pull request got closed. However, we think that if backwards compatibility was of less importance, the discussion would be pointed towards removing as much functional debt as possible.

This showed that commitment towards backwards compatibility can lead to functional debt, which can be very hard or impossible to fix over time. If, at the time of implementing, one would have noticed it would increase functional debt, the issue could have been provoked and resolved, since it would not have introduced breaking compatibility cost.

19.7.5   Concluding Remarks

There are numerous ways to prevent functional debt being introduced. Preventing functional debt is important when considering backwards compatibility. We think PowerShell can improve on this debt by focusing on the following:

- Have accurate documentation about the expected functionality of the system.
- Carefully consider introducing any new feature before shipping it.
- Provide indications how to measure backwards compatibility cost and functional debt to help in the decision making.
- Reduce technical debt so bugs are less likely to occur.

19.8   Conclusion

To quote the user @mklement0:

> PowerShell has the potential to become the lingua franca of the shell world.

This means that PowerShell has the potential to become the shell that is used by developers from different backgrounds. However, as the architecture shows, there is much to improve in order to achieve this potential.

PowerShell has:

- a good governance model and integration policy. Pull requests are thoroughly examined by those responsible for merging them.
- a module structure that could potentially use some organization. Developers might find it difficult to know where they will need to place their code at, or where to find the implementation of certain functionality.
- simple and good working common design patterns in the code.
- to pay more attention to technical debt. There is unmaintainable code due to high cyclomatic complexity, and therefore it also has low testability. This in turn reflects on the functional debt.
- an understandably strict policy on backwards compatibility. However, this causes much of the functional debt in the system to be unpayable.

All things considered, PowerShell does a relatively good job on their architecture. We hope that the future shows that it is possible to overcome the challenges which are currently present in the project.

19.9 References

Chapter 20

PyTorch

By Ziyu Bao, Tian Tian, Yuanhao Xie, Zhao Yin from TU Delft.

20.1   Abstract

PyTorch is an open-source deep learning platform. In this report, we systematically analyzed it and obtained a structural view of its architecture. Analysis include its stakeholders, context view, development view, technical debt and deployment view.

20.2   Table of Contents

20.3   Introduction

PyTorch is created and developed primarily by Facebook's artificial intelligence group as an open-source deep learning architecture that provides a seamless path from research prototyping to production deployment. Everyone could use it to build its own customized neural networks or perform fast matrix operations on GPUs using the torch component. In this report, we want to systematically analyze its architecture.

20.4   Stakeholder Analysis

In this section, we will perform a PR analysis, identify the stakeholders and other kinds of stakeholders and identify integrators of Pytorch. We also present a Power vs. Interest grid and give an analysis of the figure.

20.4.1   PR Analysis

A codification process is in the appendix.

The author of the pull request creates a new functionality and shows the usage to whom it will concern. After the showcase, two main members of PyTorch have a lot of reviews, suggestions, technical questions and vital decisions in the conversations of the pull request which are basically about, in the code level, improvements, small bugs, big issues, abstract representations (e.g., one attribute should morally be included in another class) and connections with other parts of PyTorch. This issue-solution pattern as an atom or a routine circle is repeated many times before a final approvement is reached to ensure the whole process

trackable and efficient. Experienced reviewers largely influence the enhancement and complement of the new functionality or the code to be specific.

Looking over all the conversations during which the code base of the original pull request gets improved and reconstructed, I notice that the members of PyTorch expect to see this new change thus the whole process is going quite fast and active. Meanwhile, the author is able to fully understand the suggestions, follows them strictly and comments every subtle change to make things clear, which saves quite a lot of time. With efficient and extensive cooperative efforts of the author and reviewers, the pull request becomes good enough and is naturally ready to merge.

After this kind of laborious analysis, we are quite sure what it is about and what is going on within this specific pull request. Besides, whether a pull request would be accepted depends a lot on the main developers' perspective and its external connection with the interest of the open source project.

20.4.2 Stakeholders

According to stakeholders in Chapter 9 of Rozanski and Woods [1], we classified stakeholders into 11 different types. Table below introduces each type of stakeholders in detail, including a short summary and description.

Type	Stakeholders	Summary	Description
Developers	Core developers(@fmassa,@apaszke, etc.) and contributors	Developers are the largest group of all stakeholders. Anyone involved in the software development can be seen as a developer.	@apaszke @fmassa are core developers. They are almost involved in every pull request. They are also responsible for merging the accpted pull requests. Up to now, there are 994 contributors who developed Pytorch on GitHub, including aspects like system design and code writing.
Acquirers	Organizations, research institutes	Acquirers oversee the procurement of the software and give the financial support. This kind of stakeholders usually is the most important part, because they can control the future roadmap of the software.	Facebook is the main financial supporter of PyTorch. In addition, Facebook, the Idiap Research Institute, New York University (NYU) and NEC Labs America hold copyrights of Pytorch.

[1]William Del Ra, III. 2012. Software systems architecture: second edition by Nick Rozanski and Eoin Woods. SIGSOFT Softw. Eng. Notes 37, 2 (April 2012), 36-36. DOI: https://doi.org/10.1145/2108144.2108171

Type	Stakeholders	Summary	Description
Assessors	Developers, internal assessment department and external legal entities	Assessors supervise whether the development or test of the software meet the legal regulations.	Developers, in addition to developing the system, also assess if the system is confronted to standards and legal regulations. There are also external legal departments that always check the system's legal compliance.
Communicators	Community participants, teachers	Communicators manage to explain the system to other stakeholders.	GitHub, PyTorchDiscuss and Slack are three official communities for stakeholders to ask questions and answer questions. In addition, PyTorch tutorials can be found in many webstations.
Maintainers	Some developers	Maintainers guarantee the normal operating of the system.	Some developers are in charge of the daily maintenance of Pytorch on GitHub. @ezyang is one of the maintainers. He has claimed a method to enable non-facebook employees to get their pull requests merged.
Production engineers	@Yinghai Lu and @Duc Ngo and other engineers	Production engineers are in charge of the deployment of hardware and software environments in the system.	@Yinghai Lu and @Duc Ngo are software engineers of PyTorch. They take charge of tensors and dynamic neural networks in python with strong GPU acceleration.

Type	Stakeholders	Summary	Description
Suppliers	Libraries, GitHub	Suppliers provide the hardware, software, or infrastructure which the system needs while developing.	PyTorch has a rich ecosystem of tools and libraries to support it. Alibaba (Arena): Alibaba cloud's deep learning solution-Arena supports the PyTorch frameworks. fastai, Flair, Glow, GPyTorch, Horovod, Ignite, ParlAi, PyroAi, TensorLy and Translate are all suppliers providing services for PyTorch. What's more, GitHub is the main supplier for its development.
Support staff	Developers, Teachers, Customer service	Support staff (including technical support, customer service, etc.) support the product or system when it runs.	Developers have the right to give technical support on commnuities. Teachers help users for running the system. Customer services of PyTorch provide service for solving the problems from users.
System administrators	Main engineers @Yinghai Lu and @Duc Ngo and other engineers, core developers	System administrators coordinate the overall development.	The main engineers take charge of the operation of PyTorch. Core developers control the evolution and development of different projects on PyTorch.
Testers	Developers @MlWoo,etc.	Testers systematically test the system to determine if it is suitable for deployment and use.	One of the testers is @MlWoo. He benchmarked the functionality of CNN of PyTorch on different mainstream CPU platforms

Type	Stakeholders	Summary	Description
Users	All developers and some organizations using PyTorch	Users experience and use Pytorch and have the right to give feedback. Developers also use Pytorch.	AllenNLP is an open-source research library built on PyTorch for designing and evaluating deep learning for NLP. ELF is a platform for game research that allows developers to test and train their algorithms in various game environments. Stanford University uses Pytorch on their researches of new algorithmic approaches. Udacity uses PyTorch to educate the next wave of AI innovators. Salesforce uses PyTorch to push the state of the art in NLP and Multi-task learning.

20.4.3   Going beyond Rozanski and Woods classification

Competitors: Competitors can be one of the stakeholders that concern the development of Pytorch. In this case, TensorFlow is a competitor. It is based on Theano and developed by Google. Since Pytorch releases, Pytorch obtains lots of attention and is considered to be a better version than Tensorflow.

Founders: Founders can be considered as the first developers of a software. In our project, PyTorch is created by an AI research team of Facebook. The original authors include Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan. They used the torch framework and implemented it on Python environment.

20.4.4   Integrators/Reviewers in PyTorch

The core developers such as @apaszke @fmassa are integrators. They are architects of PyTorch. Their challenges in developing Pytorch are checking all the pull requests and make sure only the pull requests which pass the integration checks can be merged. In this way, the project can keep stable. The integrators almost involve in every pull requests. They use automatic code checking to reduce the review workload. If there are mistakes in the pull requests, they ask the contributors to explain and they discuss with other integrators. The integrators are also responsible for providing more context to a particular issue if people would like to implement a feature or bug-fix for an outstanding issue and need more information.

20.4.5 Contact Persons

Pytorch has many developers who are the main members of the team. We want to contact them to better understand Pytorch. We just went to their github pages and find their email addresses if they have provided them. We also chose to leave a comment in the discussion they involved to contact them. Here is a list that we contacted: Zeming Lin, Yuandong Tian and Edward Yang.

20.4.6 Power vs Interest Grid

The figure below shows the Power & Interest grid.
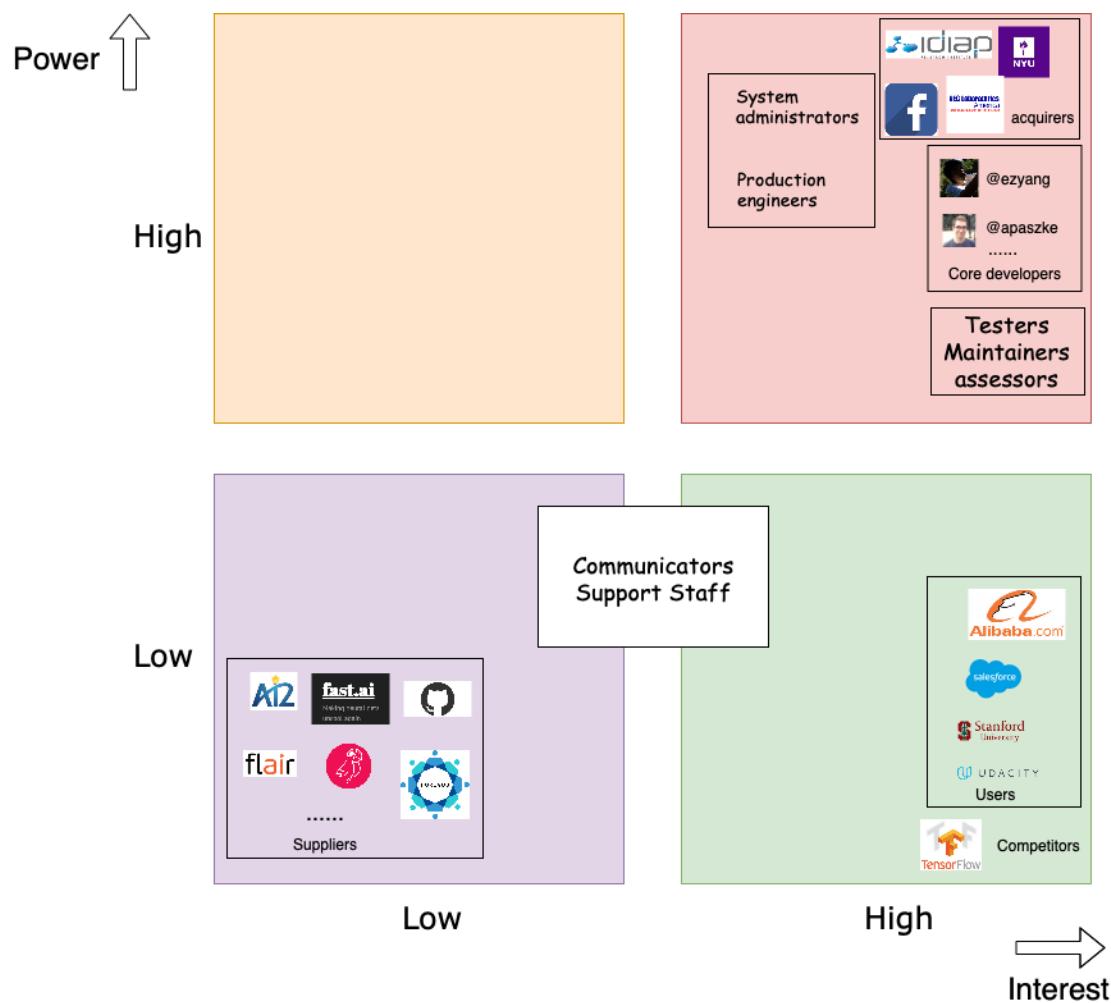
Figure 20.1: Power & Interest grid

High power and high interest: Core developers have the right to accept project modifications but not like

administrators and production engineers who have more power. Testers, maintainers, and assessors take responsibility of testing the system, managing the evolution, and overseeing the system's conformance to standard, respectively. Therefore, they have a slightly lower power compared with core developers. Testers, maintainers and assessors have the same level interest as core developers. Acquirers oversee the procurement of the system and give the financial support. Within them, Facebook is the founder. They usually are the most important part, as they can control the future roadmap of the software. Therefore, they have the highest interest and power.

Low power and high interest: Users use PyTorch and cares about its development. But they do not have high power compared with stakeholders described above. Competitors do not have power on Pytorch, but they are highly interested in every detail of Pytorch.

Low power and low interest: Suppliers have their services used by Pytorch but they do not necessarily care about Pytorch and they do not have power in Pytorch either.

Communicators and Support Staff: Communicators mainly focus on creating documentation and training material to explain PyTorch. In general, they do not have other power. Support Staff help users to run the system. They have no decision power. Therefore, both of them are mildly interested in PyTorch with relatively low power.

20.4.7   Stakeholder Analysis conclusion

In conclusion, we make a brief analysis on a pull request. We then introduce many types of stakeholders and specify what they concern and where they are involved in Pytorch. There are also two types of stakeholders beyond R&W's classification which are founders and competitors. They have different interest and power to the project. Integrators are identified above. Their challenges and merging strategy are discussed. At last, we prensent a Power&Interest of the stakeholders of Pytorch.

20.5   Context view

This session describes the scope and responsibilities of Pytorch and its relationships with external entities. The interface with external entities will be described in detail in a diagram.

20.5.1   System Scope and Responsibilities

Pytorch has its trace of development of its scope and responsibilties. In the Roadmap to Pytorch 1.0, the Pytorch team has described their thoughts on the scope of Pytorch 1.0 compared with Pytorch 0.2, 0.3 and 0.4. We can learn from their thoughts and define the scope and responsibilities of Pytorch as follows: 1. Tensor computation (like NumPy) with strong GPU acceleration 2. Deep neural networks built on a tape-based autograd system 3. Support for wide AI uses 4. Good for research and hackability 5. Support efficient industry production at massive scale 6. Support exporting models to Python-less environment 7. Support debugging for exported models 8. Support user-written C++ extensions 9. Support for platforms of Caffe2 (iOS, Android, Raspbian, Tegra, etc) and will continue to expand various platforms support

20.5.2 Context Diagram

The context diagram shows interfaces of Pytorch with external entities. The external entity can be a person, an organization or a system that "implements, offers or uses services of Pytorch, or manages, provides or uses data of Pytorch" [2]. The diagram presents the data/services transferring between external entities and Pytorch.

20.5.3 External Entities

External entities are examined in detail as follows: * Communication: Communications are mainly done in GitHub, PyTorchDiscuss and Slack. Communicators supply data to Pytorch in the form of conversational materials to be studied to help Pytorch improve.

- Storage: AWS, Google and Microsoft all have provided support of their cloud platforms for data storage of PyTorch.

- Tools: Horovod is a distributed training framework that can be used by PyTorch. Pytorch Geometry is a library of PyTorch for geometric computer vision. TensorBoardX is a visulization tool that can log events happening e.g. during training of a neural network. Translate extends PyTorch functionality to train for machine translation models.

- Users: PyTorch is used from industry to acedemia. When it is used in industry, it is used as part of the core business of companies, like Alibaba or Salesforce, to support deep learning framework. When it is used in academia, PyTorch can support researches by providing new algorithmic approaches, like in Stanford University. These users in turn also provide feedback and data to Pytorch to train with, which makes them an external entity.

20.6 Development View

The development view of PyTorch describes its code structure and dependencies, build and configuration of deliverables, system-wide design constraints and system-wide standards to ensure technical integrity [3]. We here analyze the module structure model, common design models and codeline model.

20.6.1 Module Structure Model

The main structure of PyTorch in a architectural view is shown in the figure below.

---

[2]William Del Ra, III. 2012. Software systems architecture: second edition by Nick Rozanski and Eoin Woods. SIGSOFT Softw. Eng. Notes 37, 2 (April 2012), 36-36. DOI: https://doi.org/10.1145/2108144.2108171

[3]William Del Ra, III. 2012. Software systems architecture: second edition by Nick Rozanski and Eoin Woods. SIGSOFT Softw. Eng. Notes 37, 2 (April 2012), 36-36. DOI: https://doi.org/10.1145/2108144.2108171
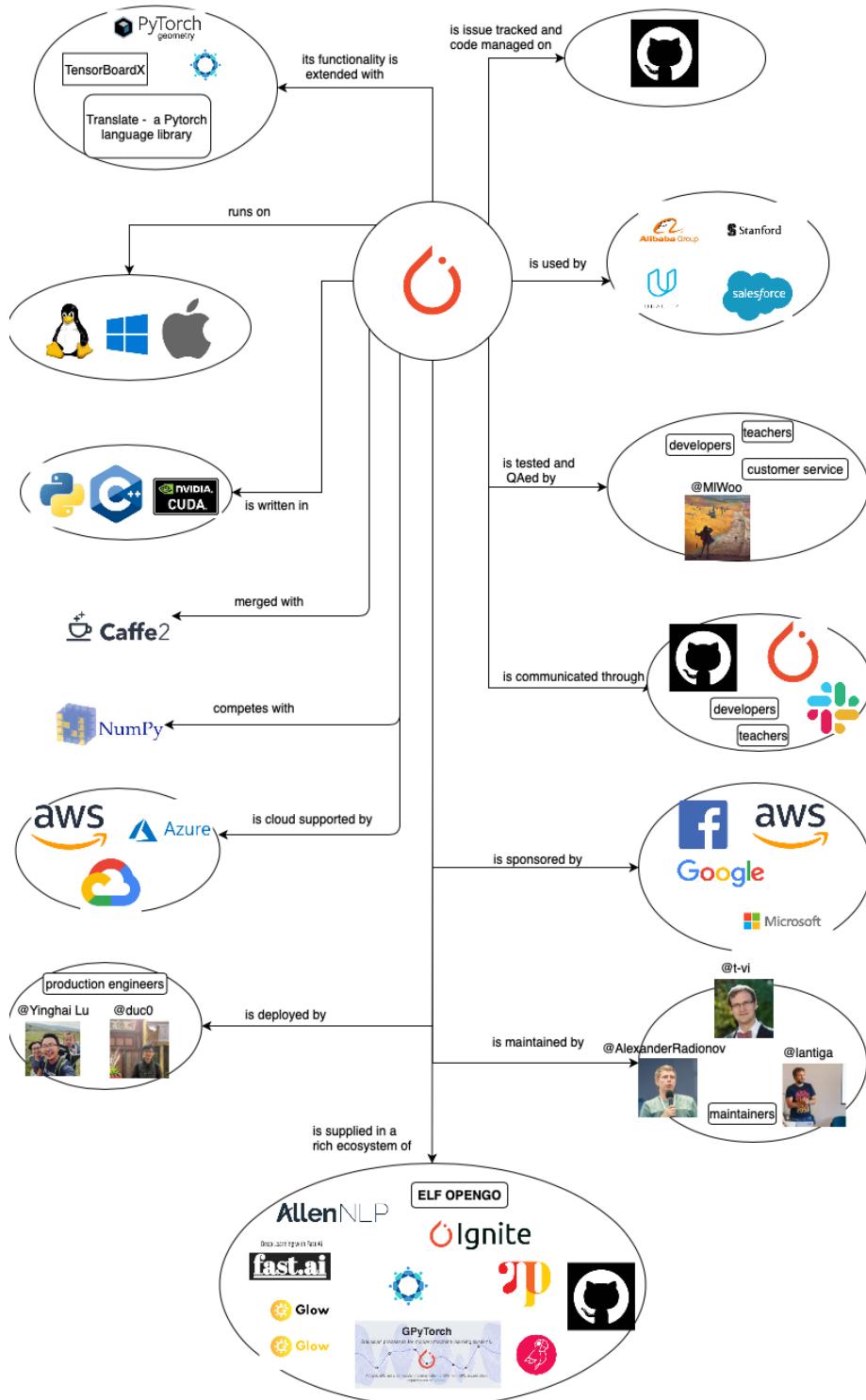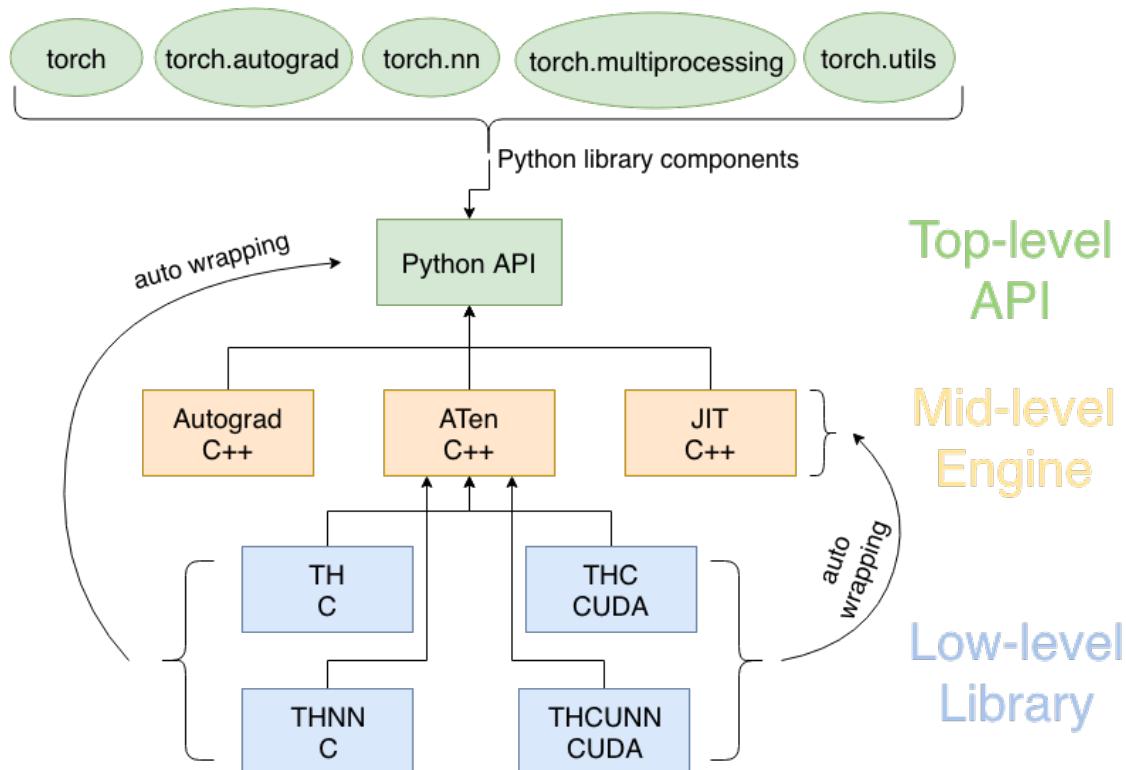
Figure 20.2: Context view

Pytorch Architecture. Inspired by [4]

The top-level Python library of PyTorch (please refer to the following section) exposes easy-to-understand API for users to quickly perform operations on tersors, build and train a deep neural network. This library provides interface but doesn't really execute the computations. Instead, it delivers this job down to its efficient computation engines written in C++.

These engines on the middle-level of module structure consist of `autograd` for computing DCG and providing auto-differentiation, `JIT` (just-in-time) compiler for optimizing computation steps that are traced, `ATen` as a C++ tensor library that wraps low-level C library for PyTorch (no autograd support).

The low-level C or CUDA library does almost all the intensive computations assigned by upper-level API. These libraries provide efficient data structures, the tensors (a.k.a. multi-dimensional arrays), for CPU and GPU (`TH` and `THC`, respectively), as well as stateless functions that implement neural network operations and kernels (`THNN` and `THCUNN`) or wrap-optimized libraries such as NVIDIA's `CuDNN` [5].

`ATen` wraps those low-level libraries in C++ and then exposes to the high-level Python API. Similarly, the neural network function libraries (low-level) are automatically wrapped towards the engine and Python API (see the two curved arrows). In other words, the low-level libraries can be utilized not only by its standard wrapper ATen but also top-level Python APIs and mid-level engines. This kind of connection keeps the code

---

[4]Deep Learning with PyTorch by Eli Stevens Luca Antiga. MEAP Publication. https://livebook.manning.com/#!/book/deep-learning-with-pytorch/welcome/v-7/

[5]Deep Learning with PyTorch by Eli Stevens Luca Antiga. MEAP Publication. https://livebook.manning.com/#!/book/deep-learning-with-pytorch/welcome/v-7/

loosely coupled, decreasing the overall complexity of the system and encouraging further development [6].

20.6.2 Component Overview

PyTorch as a libary consists of the following components (also see Figure 3 for the connection with the module structure): - **torch:** a Tensor library like `NumPy`, with strong GPU support [7]. It contains data structures for multi-dimensional tensors, and defines many mathematical operations based on these tensors. Different from its analogue `NumPy`, all the data structures and tensor operations can be seamlessly performed from CPU to GPU which would accelerate the computation by a huge amount. - **torch.autograd:** a tape-based automatic differentiation library that supports all differentiable Tensor operations in `torch` [8]. This functionality greatly differs PyTorch from other machine learning or deep learning frameworks like TensorFlow [9], Caffe [10] and CNTK [11] which require users to restart from scratch at beginning in order to change some minor behaviors of the neural network once created. While in PyTorch, a technique called reverse-mode auto-differentiation is adopted to facilitate differentiation process so that the computation graph is computed in the fly which leaves users more time to implement their ideas. - **torch.nn:** a neural networks library deeply integrated with autograd designed for maximum flexibility [12]. This component or module in PyTorch provides a high-level functionality for us to build and train a deep neural network easily without pain. It contains many types of neural network layers like convolutional layers, recurrent layers, padding layers and normalization layers. - **torch.multiprocessing:** Python multiprocessing, but with magical memory sharing of torch Tensors across processes. Useful for data loading and Hogwild training [13]. This component wraps native Python multiprocessing module using shared memory to provide shared views on the same data in different processes. - **torch.utils:** DataLoader, Trainer and other utility functions for convenience [14]. It consists of five submodules - `torch.utils.bottleneck` for debugging bottlenecks in the program, `torch.utils.checkpoint` for checkpointing a model or part of the model and etc.

20.6.3 Common Design Model

This section uses a common design model to analyze how PyTorch tries to achieve its developmental approach. #### Common processing

Common processing identifies tasks that benefit greatly from using a standard approach across all system elements.

---

[6]Deep Learning with PyTorch by Eli Stevens Luca Antiga. MEAP Publication. https://livebook.manning.com/#!/book/deep-learning-with-pytorch/welcome/v-7/

[7]PyTorch github website, https://github.com/pytorch/pytorch.

[8]PyTorch github website, https://github.com/pytorch/pytorch.

[9]Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org

[10]Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv:1408.5093>arXiv:1408.5093* 2014

[11]Seide, Frank & Agarwal, Amit. (2016). CNTK: Microsoft's Open-Source Deep-Learning Toolkit. 2135-2135. 10.1145/2939672.2945397.

[12]PyTorch github website, https://github.com/pytorch/pytorch.

[13]PyTorch github website, https://github.com/pytorch/pytorch.

[14]PyTorch github website, https://github.com/pytorch/pytorch.

Use of third-party libraries. PyTorch makes use of third-party libraries for build, doc generation, high-level operations and etc. Some of them are: - **python-future:** a missing compatibility layer between Python 2 and Python 3 [15]. - **numpy:** a fundamental package needed for scientific computing with Python [16]. - **pyyaml:** a next generation YAML parser and emitter for Python [17]. - **setuptools:** a fully-featured, actively-maintained, and stable library designed to facilitate packaging Python projects [18]. - **six:** a Python 2 and 3 compatibility library that provides utility functions for smoothing over the differences between the Python versions with the goal of writing Python code that is compatible on both Python versions [19]. - **typing:** a Python library support for type hints. - **matplotlib:** a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms [20].

Documentation management. Well documented code benefits code readability, practical implementation and issue tracing. PyTorch uses Google style for formatting docstrings. Length of line inside docstrings block must be limited to 80 characters to fit into Jupyter documentation popups. For C++ documentation it uses Doxygen and then convert it to Sphinx via Breathe and Exhale [21].

20.6.3.1 Standard design approaches

Standard design approaches identifies how to deal with situations where implementations of a certain aspect of an element will have a system-wide impact.

API design standardization. (resource from [22]) The essential unit of PyTorch is the Tensor. As the Tensor is used in Python for users but implemented in C, it needs API design to wrap it up so that user can interact with it from the Python shell. PyTorch first extends the Python interpreter by CPython's framework to define a Tensor type that can be manipulated from Python. The formula for defining a new type is as follows: 1) create a struct that defines what the new object will contain and 2) define the type object for the type. Then PyTorch wraps the C libraries that actually define the Tensor's properties and methods by CPython backend's conventions. The `Tensor.cpp` file defines a "generic" Tensor and PyTorch uses it to generate Python objects for all the permutations of types (`float`, `integer`, `double` and etc.). PyTorch takes the custom YAML-formatted code and generates source code for each method by processing it through a series of steps using a number of plugins. Finally to generate a workable application, PyTorch takes a bunch of source/header files, libraries, and compilation directives to build an extension using Setuptools.

Contributing standardization. As PyTorch is an open source, everyone is free to contribute to the repository. In order to keep maintainability, reliability, and technical cohesion of the system, the PyTorch Governance (consisting of code maintainers, core developers and some other contributors) composed a contributing tutorial to standardize the design process. The tutorial provides useful guidelines for coding, parameter configuration, testing, writing documentation and all other tips and rules for qualified contribution.

---

[15] Python-future github website. https://github.com/PythonCharmers/python-future
[16] Numpy github website. https://github.com/numpy/numpy
[17] Pyyaml github website. https://github.com/yaml/pyyaml
[18] Setuptools documentation. https://setuptools.readthedocs.io/en/latest/
[19] Six github website. https://github.com/benjaminp/six
[20] Matplotlib github website. https://github.com/matplotlib/matplotlib
[21] PyTorch github website, https://github.com/pytorch/pytorch.
[22] PyTorch blog. https://pytorch.org/blog/a-tour-of-pytorch-internals-1/

20.6.4   Codeline Model

20.6.4.1   Source Code Structure

PyTorch has its code structure to make it easy for developers to locate the code they want to change. We show its main directory below while the full directory could be seen on here.

```
Pytorch Code Structure
├── c10/ – Core library files that work everywhere.
├── aten/ – C++ tensor library for PyTorch (no autograd support).
│   └── src/
│       ├── TH/ THC/ THNN/ THCUNN/ – Legacy library code from the
│       │   original Torch.
│       └── ATen/
│           ├── core/ – Core functionality of ATen.
│           └── native/ – Modern implementations of operators.
│               ├── cpu/ – CPU implementations (compiled with processor
│               │   -specific instructions) of operators.
│               ├── cuda/ – CUDA implementations of operators.
│               ├── sparse/ – CPU and CUDA implementations of COO sparse
│               │   tensor operations.
│               └── mkl/ mkldnn/ miopen/ cudnn/ – Implementations of
│                   operators which simply bind to some backend library.
├── torch/ – The actual PyTorch library.
│   └── csrc/ – C++ files composing the PyTorch library.
│       ├── jit/ – Compiler and frontend for TorchScript JIT frontend.
│       ├── autograd/ – Implementation of reverse-mode automatic
│       │   differentiation.
│       ├── api/ – The PyTorch C++ frontend.
│       └── distributed/ – Distributed training support for PyTorch.
├── tools/ – Code generation scripts for the PyTorch library.
├── test/ – Python unit tests for PyTorch Python frontend.
│   ├── test_torch.py – Basic tests for PyTorch functionality.
│   ├── test_autograd.py – Tests for non-NN automatic differentiation
│   │   support.
│   ├── test_nn.py – Tests for NN operators and their automatic
│   │   differentiation.
│   ├── test_jit.py – Tests for the JIT compiler and TorchScript.
│   ├── cpp/ – C++ unit tests for PyTorch C++ frontend.
│   ├── expect/ – Automatically generated "expect" files which are used
│   │   to compare against expected output.
│   └── onnx/ – Tests for ONNX export functionality, using both PyTorch
│       and Caffe2.
└── caffe2/ – The Caffe2 library.
    ├── core – Core files of Caffe2, e.g., tensor, workspace, blobs, etc.
    ├── operators – Operators of Caffe2.
    └── python – Python bindings to Caffe2.
```

20.6.4.2   Build approach

For development build, PyTorch and its third-party frameworks are built using Python Setup tools. We need to specify the command parameters if only part of the components are required to rebuild Pytorch. For users build, both package managers - Anaconda and pip could be used to build the whole project.

20.6.4.3   Release process

PyTorch has released 21 versions since 2016. We divided them into three stages. At the beginning, Pytorch was busy with developing new functions, so it released a new version every month, or even twice a month. In the second stage, Pytorch tended to be stable, so it released nearly every two months. In the recent stage, Pytorch focused more on fixing existing bugs than developing new functions, so it released at a frequency lower than before.

20.7   Technical Debt

Technical debt reflects the implicit cost of choosing a simple solution instead of using a better method that takes a longer time. In this section, we first identify different forms of technical debts which include the keywords/tags, the complexity trend of the hotspot, code smell(temporal coupling and duplicated code), and bugs. We also list the methods to reduce or avoid these technical debts. Then, we discuss the testing debt. In the end, the evolution of technical debt is explained. ### Identifying Technical debt

The tools we used to identify the technical debts are listed below:

Name of tool	Usage
CodeScene	Identifying hotspot, complexity trends of the hotspot, and temporal coupling
Github-grep	Finding keywords like `TODO`, `FIXME` and `XXX`
SonarQube	Identifying duplications and finding bugs

The reasons we decided to use these tools are: 1. **CodeScene** is an easily operated tool and has several intuitively functions to show the code structure and point out some parts that really matter, for example, the hotspots. We choose three functions to identify hotspot, complexity trends and temporal coupling. 2. **Github-grep** is a tool of Git to search everything you want in the code. We used this tool to find all three keywords: `TODO`, `FIXME` and `XXX`. 3. **SonarQube** is a powerful code quality management tool. It detects the code quality in the following aspects: bugs, vulnerabilities, code smells, code coverage and duplications. Since some of those aspects we have used `codescene` to analyze, we mainly analyze the duplications and the bugs by using SonarQube.

20.7.0.1 Keywords/tags

Keywords/tags indicates the technical debts which the developer noticed but did not fix. As along with the accumulation of tags, some of them may be forgotten, or become bugs [23]. Furthermore, these keywords may clutter the code and have negative effects on code comprehension [24]. This large amount of technical debts may make developers difficult to comprehend the code.

We analyzed three kinds of keywords/tags "TODO"s, "FIXME"s and "XXX"s in PyTorch. We found that even though the tags have different names, but their contents are quite similar. We found 1098 "TODO"s. They are almost everywhere in PyTorch. 432 "TODO"s are in caffe2. There are 118 "FIXME"s, they are mainly in aten, caffe2, test, torch. There are 119 "XXX"s identified, 72 of them are in the main PyTorch component-torch. Some of the tags mentions who will fix it, none of them mentions when they will be fixed. Following are the examples:

- `caffe2/python/caffe_translator.py:# TODO(jiayq): find a protobuf that uses this and verify.`

- `caffe2/operators/lp_pool_op.cc:// TODO: reduce the apparent redundancy of all the code below.`

- `tools/autograd/gen_variable_type.py: # FIXME: figure out a better way when we support sparse tensors in jit`

- `caffe2/operators/conv_op_cudnn.cc:// TODO(Yangqing): a lot of the function contents are very similar. Consider`

- `torch/csrc/autograd/functions/accumulate_grad.cpp: // XXX: this method is not thread-safe!`

20.7.0.2 Complexity trend of hotspots

The reason why we analyze the hotspots is that the code with the high technical debt can be found by analyzing the 'hotspot' [25]. Hotspots are those large, complex code which the developers have to work often with. We used `Codescene` to analyze the hotspots.

As shown in the figure below, each blue circle represents a package in the code, and the red dots are the hotspots [26].

---

[23]Margaret-Anne Storey, Jody Ryall, R. Ian Bull, Del Myers, Janice Singer, "TODO or To Bug:Exploring How Task Annotations Play a Role in the Work Practices of Software Developers", InProceedings of the 30th Int'l. Conf. Software Engineering (ICSE '08), pp. 251-260, 2008.

[24]Margaret-Anne Storey, Jody Ryall, R. Ian Bull, Del Myers, Janice Singer, "TODO or To Bug:Exploring How Task Annotations Play a Role in the Work Practices of Software Developers", InProceedings of the 30th Int'l. Conf. Software Engineering (ICSE '08), pp. 251-260, 2008.

[25]A Crystal Ball to Prioritise Technical Debt in Monoliths or Microservices: Adam Tornhill's Thoughts. https://www.infoq.com/news/2017/04/tornhill-prioritise-tech-debt

[26]A Crystal Ball to Prioritise Technical Debt in Monoliths or Microservices: Adam Tornhill's Thoughts. https://www.infoq.com/news/2017/04/tornhill-prioritise-tech-debt
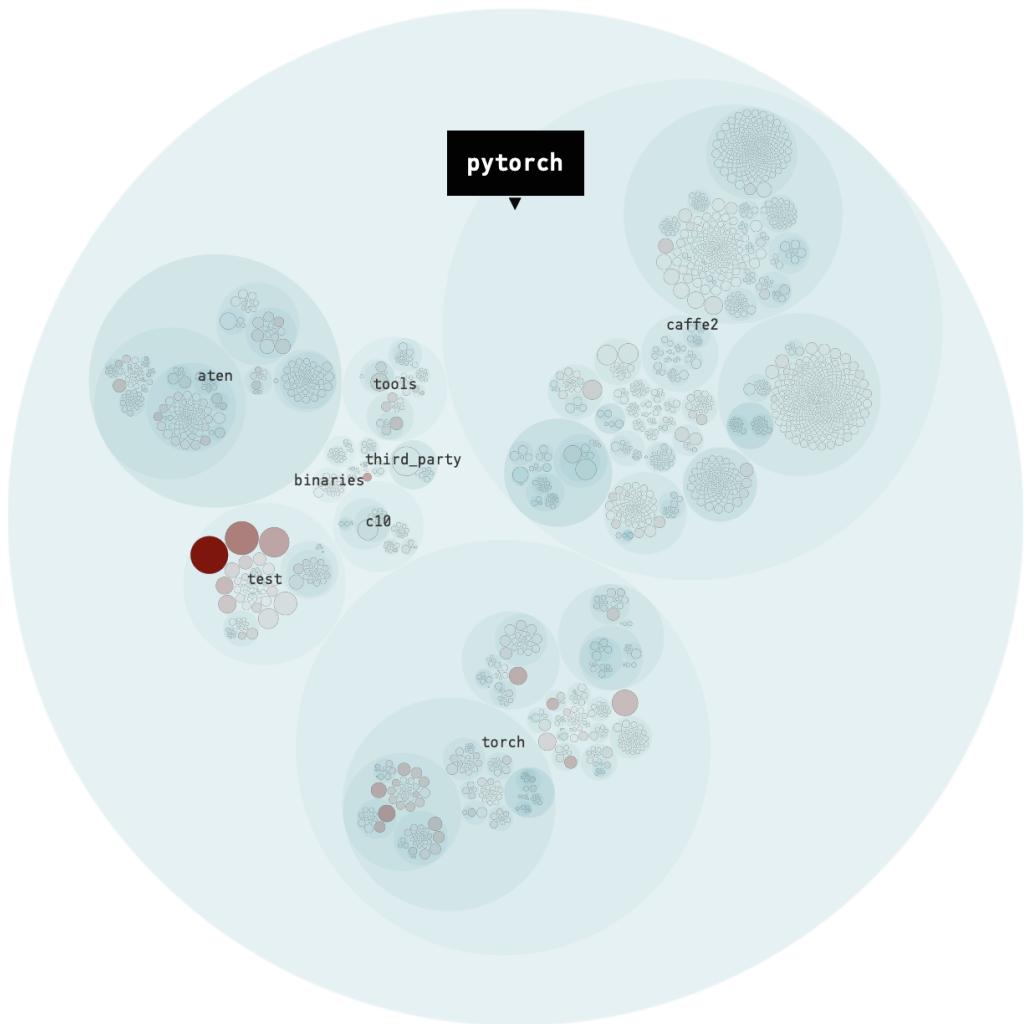
Figure 20.3: Hotspot

Complexity trends denote how do the hotspots get more complicated over time. The hotspot with fast-growing complexity should be prioritized for refactoring to reduces the risk of technical debt. Otherwise, as soon as the complexity becomes unmanageable, the code modifications become really difficult [27].

As shown in the figure below, the complexity trend of hotspot `pytorch/test/test_jit.py` grows rapidly since June 2018. This accumulating complexity is a sign that the hotspot needs refactoring [28] to reduce the risk of technical debt.
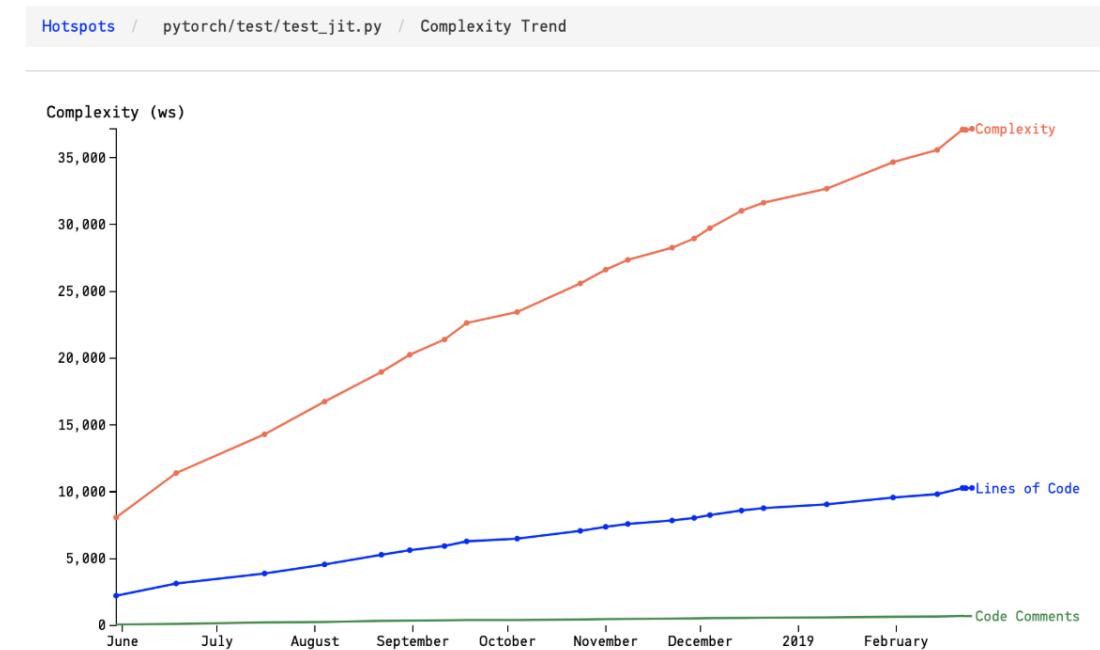
Figure 20.4: Complexity trends of pytorch/test/test_jit.py

20.7.0.3 Temporal coupling

Temporal coupling is a kind of codesmell. The large codebases with multiple developers may lead to technical debt. Even though at the beginning of development, developers separate the modules on purpose, the couplings may still form during development. The couplings among different modules lead to rigid system design and the difficulty of extending the features [29]. Temporal coupling indicates how two or more modules change together over time. It can be obtained by calculating how often a module changes together with other modules. We used `Codescene` to visualize the temporal coupling.

In the figure below, the lines that the modules which are modified together. The thicker lines, the stronger of temporal coupling.

---

[27]A Crystal Ball to Prioritise Technical Debt in Monoliths or Microservices: Adam Tornhill's Thoughts. https://www.infoq.com/news/2017/04/tornhill-prioritise-tech-debt

[28]Codescene Guides-COMPLEXITY TRENDS https://codescene.io/docs/guides/technical/complexity-trends.html

[29]Coupling analysis https://github.com/smontanari/code-forensics/wiki/Coupling-analysis
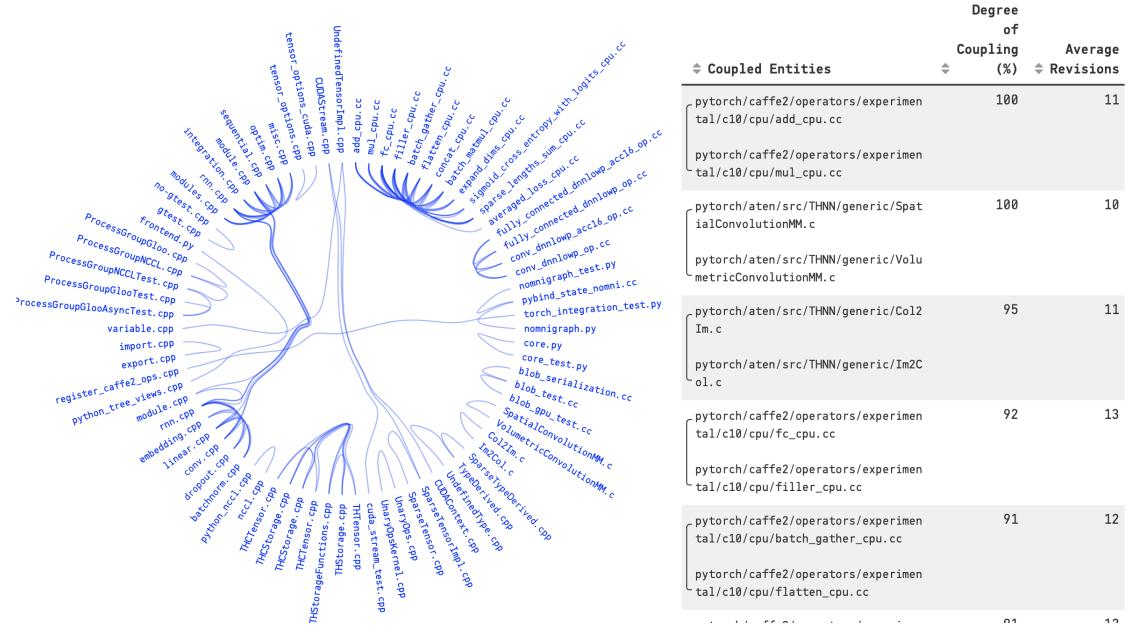
Figure 20.5: Temporal coupling

The coupling degrees of the pairs shown in the table in the right of the Figure are quite strong. For example, the coupling degrees of the third pairs are 95%. It means that 95% chance of changing one file results in a change in another file. The temporal coupling sometimes suggests a refactoring. As you can see the coupling degrees of the first pairs are 100%. The figure below shows the code of `add_cpu.cc` and `mul_cpu.cc`. These two files are more like copy-paste work.

20.7.0.4 Duplications and existing bugs

To identify duplications and existing bugs, we utilized `SonarQube`.
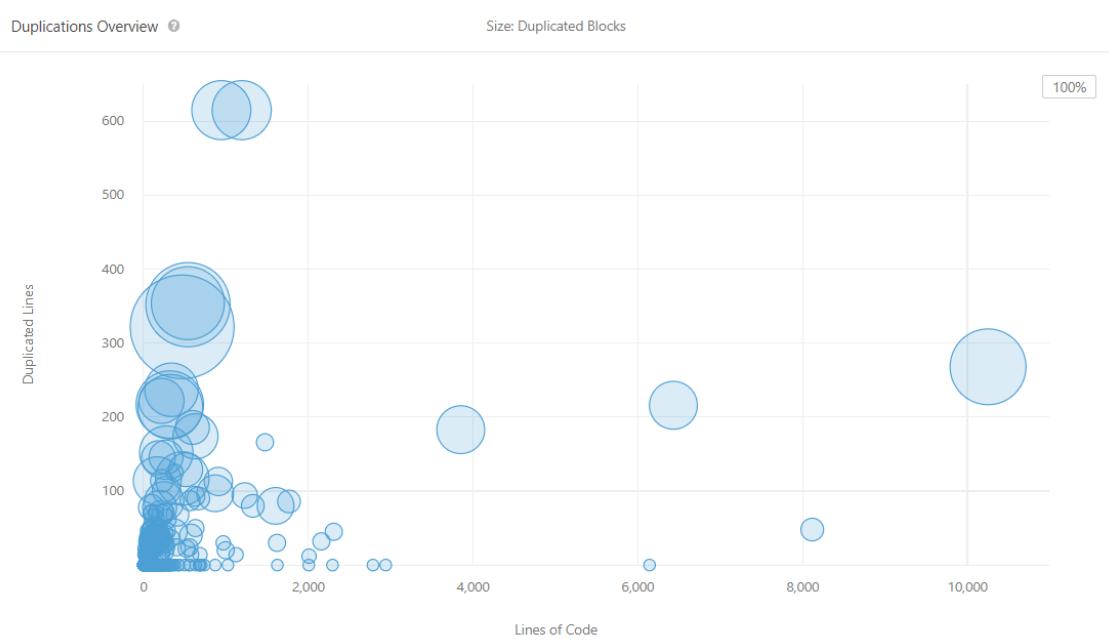
Duplicates: Duplicates illustrate those recurring code. It belongs to code smell which is the potential threat in software development. Developers should keep less duplicated code to make the code clean. The figure below shows all the duplicates in PyTorch. There are overall 3.6% duplicates. It should be noticed that the test files contain the most duplicates. In our analysis, test files need to test different cell but theoretically use the same way, so the codes look like the same.

```
43        static_cast<int>(pre), static_cast<int>(n), static_cast<int>(post)};
44      B_dims = {static_cast<int>(n), 1};
45    }
46  } else {
47    std::copy(A.sizes().cbegin(), A.sizes().cend(), std::back_inserter(A_dims));
48    std::copy(B.sizes().cbegin(), B.sizes().cend(), std::back_inserter(B_dims));
49    const std::vector<int> C_dims =
50      caffe2::elementwise_ops_utils::ComputeBinaryBroadcastForwardDims(
51        A_dims, B_dims);
52    if (A.is_same(C)) {
53      CAFFE_ENFORCE_EQ(C_dims, A_dims);
54    } else if (B.is_same(C)) {
55      CAFFE_ENFORCE_EQ(C_dims, B_dims);
56    } else {
57      C.Resize(C_dims);
58    }
59  }
60  auto* C_data = C.template mutable_data<DataType>();
61
62  caffe2::math::Add(
63      A_dims.size(),
64      A_dims.data(),
65      B_dims.size(),
66      B_dims.data(),
67      A.data<DataType>(),
68      B.data<DataType>(),
69      C.mutable_data<DataType>(),
70      static_cast<CPUContext*>(&context));
71  }
72  } // namespace
73  } // namespace caffe2
74
75  namespace c10 {
76  C10_REGISTER_KERNEL(caffe2::ops::Add)
77      .kernel<decltype(caffe2::add_op_cpu_impl<float>), &caffe2::add_op_cpu_impl<floa
78      .dispatchKey(CPUTensorId());
79  } // namespace c10
```

```
44        static_cast<int>(pre), static_cast<int>(n), static_cast<int>(post)};
45      B_dims = {static_cast<int>(n), 1};
46    }
47  } else {
48    std::copy(A.sizes().cbegin(), A.sizes().cend(), std::back_inserter(A_dims));
49    std::copy(B.sizes().cbegin(), B.sizes().cend(), std::back_inserter(B_dims));
50    const std::vector<int> C_dims =
51      caffe2::elementwise_ops_utils::ComputeBinaryBroadcastForwardDims(
52        A_dims, B_dims);
53    if (A.is_same(C)) {
54      CAFFE_ENFORCE_EQ(C_dims, A_dims);
55    } else if (B.is_same(C)) {
56      CAFFE_ENFORCE_EQ(C_dims, B_dims);
57    } else {
58      C.Resize(C_dims);
59    }
60  }
61  auto* C_data = C.template mutable_data<DataType>();
62
63  caffe2::math::Mul(
64      A_dims.size(),
65      A_dims.data(),
66      B_dims.size(),
67      B_dims.data(),
68      A.data<DataType>(),
69      B.data<DataType>(),
70      C.mutable_data<DataType>(),
71      static_cast<CPUContext*>(&context));
72  }
73  } // namespace
74  } // namespace caffe2
75
76  namespace c10 {
77  C10_REGISTER_KERNEL(caffe2::ops::Mul)
78      .kernel<decltype(caffe2::mul_op_cpu_impl<float>), &caffe2::mul_op_cpu_impl<floa
79      .dispatchKey(CPUTensorId());
80  } // namespace c10
```

Figure 20.6: Code of add_cpu.cc and mul_cpu.cc

Bugs: We also used `SonarQube` to find bugs in Pytorch. Bugs can lead to a really bad impact on system and developers should try their best to avoid creating bugs. In our cases, the tool detected 98 bugs in the current

release. It is shown in the first figure below. The bugs identifed by `SonarQube` indicate almost the same problem as shown in the second figure below: Those codes contain the identical sub-expressions on both sides of operator "or". This kind of codes were recognized as bugs. It should be noticed that those codes were written 2 years ago, which can be considered as a long-time debt. Those codes should be fixed for better developing.

Figure 20.7: Bugs in documents
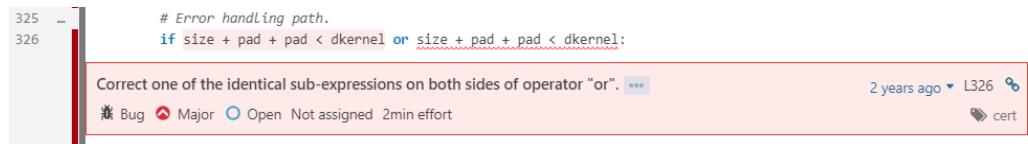
Figure 20.8: Specific bugs

20.7.0.5   Overview of the solutions

After the analysis, we came up with the solutions to reduce or solve the technical debts:

- To reduce the technique debt casued by accumulation of the keywords, the progress of fixing the

issues should be tracked. This can be done by appointing an issue tracker and ask developers to add more details in the keywords/tags, such as the name who is responsible for the issue, deadline.

- The complexity trend of the hotspots should be tracked regularly, If the complexity trend increases sharply, the hotspot should be appropriately redesigned and refactored.

- The code smell such as the temporal coupling and the duplicated code should be track. Extract the common part into a module remove the duplications.

20.7.1   Identifying Testing Debt

5% files, 32% lines covered	
Element	Statistics, %
.circleci	0% files, not covered
.ctags.d	
.github	
.idea	
.jenkins	0% files, not covered
aten	0% files, not covered
benchmarks	0% files, not covered
binaries	0% files, not covered
build	0% files, not covered
c10	
caffe2	0% files, not covered
cmake	
docker	
docs	0% files, not covered
modules	0% files, not covered
scripts	0% files, not covered
submodules	
test	3% files, 31% lines covered
third_party	0% files, not covered
tools	0% files, not covered
torch	20% files, 32% lines covered

Figure 20.9: Testing coverage

Testing debt exists due to the lack of testing or poor quality of testing. In this section, we analyze the testing debt of PyTorch.

By using a Python IDE - PyCharm, we get a detailed test coverage showing the files coverage of the whole project and lines coverage of every single Python file (see the figure above). Only 5% files and 32% lines are covered which are quite low. The reasons are two folds. One is that the tests (such as GPU test and distribution test) we run are not complete and the other one is that files in directories (e.g., `aten`, `c10`) are in C/C++ which could not be included.

Despite these exceptions, the test coverage is still not enough for such a widely used open source project. We can see that the torch module is coveraged due to it is the core module that currently being used frequently. Caffe2 and some other modules are not covered since they are rather complicated or too old, developers did not concern more on those modules. Thus, those files that not being test should be seen as test debt.

20.7.2   The evolution of Technical Debt

Since Pytorch was the next generation products of torch, it was developed officially since 2016. Pytorch was mainly developed on Github to control its different releases. As shown in the figure below, the contributions increased dramatically since 2016.
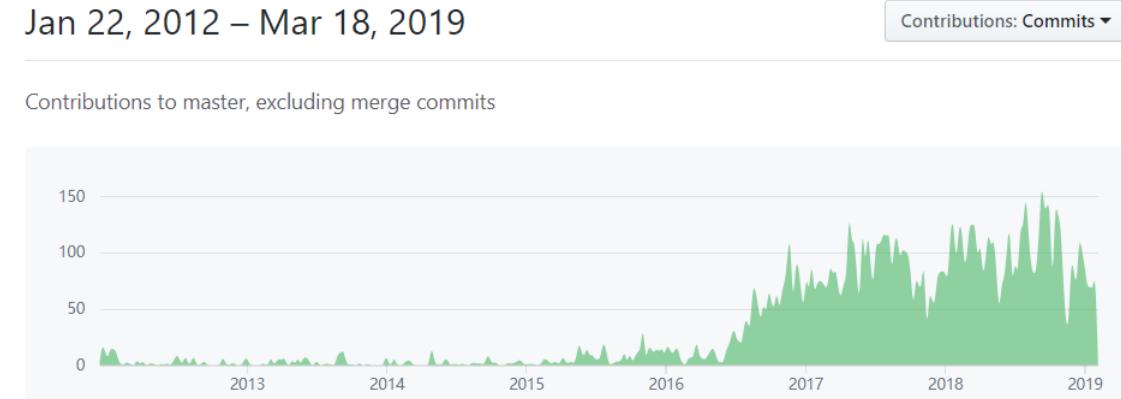
Figure 20.10: Contributions to master, excluding merge commits

According to the releases showing on the Github, Pytorch has released several versions which from v0.1.1 to v1.0.1. We summarized those versions fixing bugs into a table. We can see the evolution of technical debt from this table. At the first six versions, Pytorch focus on developing new functions regardless of fixing bugs, we considered this stage as developing stage. Then, when the system gradually keep in stable, the developers begin bugs fixing. In the recent versions, since the early versions accumulated more technical debt and gradually effect the system daily running, developers focus on fixing bugs and check errors.

Version	Bug Fix	Time
v0.1.1	alpha-1 version, did not mention bugfix	Aug 24, 2016
v0.1.2	alpha-2 version, did not mention bugfix	Sep 1, 2016
v0.1.3	alpha-3 version, did not mention bugfix	Sep 16, 2016
v0.1.4	alpha-4 version, did not mention bugfix	Oct 3, 2016
v0.1.5	alpha-5 version, did not mention bugfix	Nov 8, 2016
v0.1.6	beta version, did not mention bugfix	Jan 21, 2017
v0.1.7	A bugfix release with small features: fixed 13 bugs	Jan 26, 2017
v0.1.8	A bugfix release with small features: fixed 10 bugs	Feb 3, 2017
v0.1.9	fixed 7 bugs	Feb 17, 2017

Version	Bug Fix	Time
v0.1.10	fixed 19 bugs	Mar 5, 2017
v0.1.11	Bug fixed: torch: 6, autograd,nn,optim:13, other bugs: 2	May 1, 2017
v0.1.12	fixed 24 bugs	May 3, 2017
v0.2.0	fixed 27 bugs	Aug 28, 2017
v0.3.0	Bug fixed: torch: 1, tensor: 26, sparse: 3 autograd: 7, nn: 19, optim: 1	Dec 4, 2017
v0.3.1	Bug fixed: torch operators: 9, data: 8, autograd: 1, nn layers: 6, CUDA: 10, CPU: 1	Feb 9, 2018
v0.4.0	Bug fixed: torch operators: 18, core: 7, autograd: 3, nn layers: 11, CUDA: 10	May 30, 2018
v0.4.1	fix 25 bugs	Jul 26, 2018
v1.0rc1	serious bugs: 5, correctness bugs: 13, error check: 7	Oct 2, 2018
v1.0.0	serious bugs: 11, correctness bugs: 29, error check: 13	Dec 7, 2018
v1.0.1	serious bugs: 7, correctness bugs: 5, crashes bugs: 13	Feb 7, 2019

20.8 Deployment View

Deployment view determines the related environment to run the system. We will discuss the deployment view in the following passages and the figure below is the overall figure of the deployment view of Pytorch.

20.8.1 Third Party Library

Pytorch uses different libraries to develop its system. Those third-party libraries have been specifically introduced in section Development View. Those third parties including `Numpy`, `Sphinx`, `pyyaml`, `CuDNN`, `MKL` etc., form third-party system requirements for running Pytorch and support the daily operating of Pytorch.

20.8.2 Runtime platforms

Pytorch does not want to give up a Python frontend but a Python frontend cannot deal with, e.g., a multithreaded environment, so Pytorch employs a C++ frontend but it mimics a Python frontend closely. Pytorch has several backend modules intead of one. The modules rely heavily on linear algebra libraries like `MKL` for CPU and deep neural network libraries like `CuDNN` for GPU. Pytorch requires a 64-bit CPU. An Intel CPU is preferred because `MKL` is tuned for an Intel architecture. To benefit from GPU acceleration,
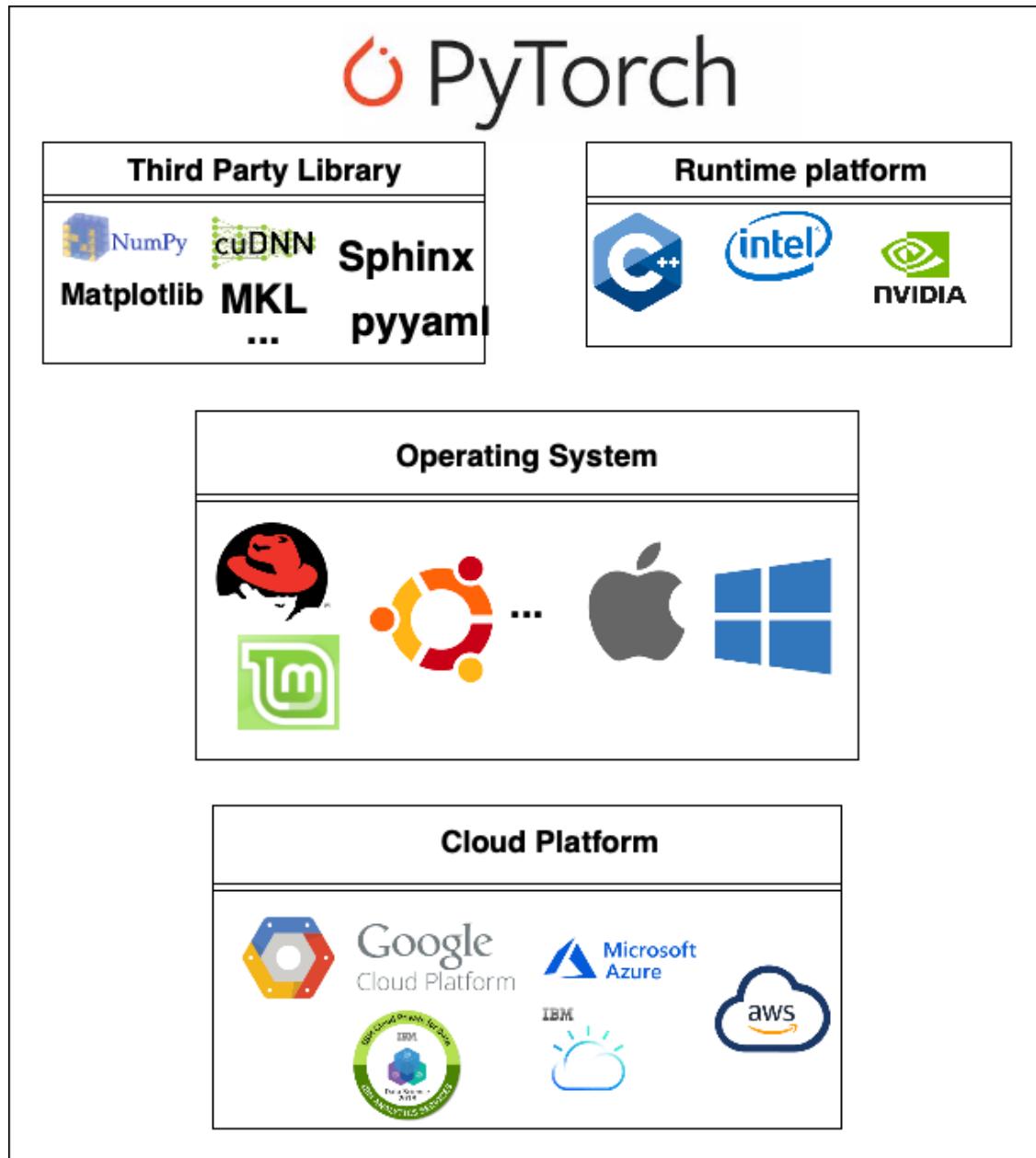
Figure 20.11: Figure deployment view

Pytorch only works on NVIDIA GPUs, because it requires CUDA support. For high-demanding tasks in research or production, it is suggested to use an HPC platform or a cloud platform, e.g. AWS.

20.8.3 Operating systems

PyTorch can be installed and used in many types of operating systems. 1. **Linux**: PyTorch 1.0 supports various Linux distributions that use glibc >= v2.17: - Arch Linux, minimum version 2012-07-15 - CentOS, minimum version 7.3-1611Debian, minimum version 8.0 - Fedora, minimum version 24 - Mint, minimum version 14 - OpenSUSE, minimum version 42.1 - PCLinuxOS, minimum version 2014.7 - Slackware, minimum version 14.2 - Ubuntu, minimum version 13.04 2. **MacOS**: PyTorch is supported on macOS 10.10 (Yosemite) or above. As MacOS does not has CUDA support building from binaries, it will not benefit from GPU accelaration. 3. **Windows**: PyTorch is supported on the following Windows distributions: - Windows 7 and greater; Recommended: Windows 10 and greater. - Windows Server 2008 r2 and greater.

20.8.4 PyTorch on cloud

Cloud platforms provide powerful hardware and infrastructure for training and testing the PyTorch models. PyTorch offers following cloud installation options:

Could Platform	Details
Microsoft Azure	Can choose between a GPU-based or a CPU-only instance of the Data Science Virtual Machine
Google Cloud Platform	On machine instances with access to one, four, or eight GPUs in specific regions or on containerized GPU-backed Jupyter notebooks
Amazon Cloud Platform	AWS Deep Learning AMI with optional NVIDIA GPU support
IBM Cloud Kubernetes cluster	On Kubernetes clusters on IBM Cloud
IBM Cloud data science and data management	Python environment with Jupyter and Spark

20.9 Conclusion

With two months of analysis of Pytorch, we gain more insights of the whole Pytorch architecture. We analyze different perspectives and viewpoints of the project to understand more about the inner workings. Even though Pytorch is a recently-developed software, it already has a well-organized architecture. And we believe it will keep developing and develop more functions for deep learning frameworks.

20.10 References

20.11 Appendix

20.11.1 PR Analysis

20.11.1.1 Pull requests which are accepted

Pull request	Lifetime	Components it touches	PR content and Related issues	Deprecate another pull request or not
Implement adaptive softmax awaiting response #5287 by @elanmart	After v0.4.0	`torch.nn`(neural networks library which is integrated with autograd designed)	This pr implements "Efficient softmax approximation for GPUs" which discussed in another pr #4659	No
Implement adaptive softmax awaiting response #5287 by @elanmart	After v0.4.0	`torch.nn`(a neural networks library deeply integrated with autograd designed for maximum flexibility)	This pr implements "Efficient softmax approximation for GPUs" which discussed in another pr #4659	No
Parallelize elementwise operation with openmp #2764 by @MlWoo	After v0.4.0	Low-level tensor libraries for PyTorch(TorcH)	This pr parallelizes elementwise operation of discontiguous THTensor with openmp	No
Implement sum over multiple dimensions #6152 by@t-vi	After v0.3.1	ATen C++ bindings	This pr implements summing over multiple dimensions as an ATen native function and fixed #2006	No

Pull request	Lifetime	Components it touches	PR content and Related issues	Deprecate another pull request or not
Fixed non-determinate preprocessing on DataLoader #4640 by @AlexanderRadionov	After v0.3.1	`torch.utils` `Dat-` `aLoader(Trainer` and other utility functions for convenience)	This pr adds ind_worker_queue parameter in DataLoader to solve the non-deterministic issue which happens when DataLoader is in multiprocessing mode	No
Introduce scopes during tracing #3016 by @lantiga	After v0.2.0	`Scope`, `IR` (intermediate representation) and `Tracing`	This pr introduced the scopes for group operations in the tracing IR	No

20.11.1.2 Pull requests which are rejected

Pull request	Lifetime	Components it touches	PR content and Related issues	Deprecate another pull request or not
Fixes errors #10765 by @peterjc123	After v0.4.0	ATen C++ bindings	Fixes errors when linking caffe2 statically related to issues #10746 and [#10902](https://github.com/pytorch/pytorch/issues/10902	No
Low rank multivariate normal #8635	After v0.4.0	`torch.distributions` `torch.trtrs`	Implements low rank multivariate normal distribution where the covariance matrix has the from `W @ W.T + D`.	No
[caffe2]seperate mkl, mklml, and mkldnn #12170	After v0.4.1	Caffe2 component of PyTorch and Docs.	1. Remove avx2 support in mkldnn. 2. Seperate mkl, mklml, and mkldnn. 3. Fix convfusion test case	No

Pull request	Lifetime	Components it touches	PR content and Related issues	Deprecate another pull request or not
Checkpointing #4594	After v0.3.0	`torch.autograd`, `torch.utils` and Docs.	Autograd container for trading compute for memory.	Be deprecated by pull request #6467.
Adding katex rendering of equations #8848	After v0.4.0	Docs, `torch/functional` and `torch.nn`	This fixes issue #8529. 1. Adds Katex extension to `conf.py` and requirements.txt. 2. Fixes syntax differences in docs. 3. Should allow documentation pages to render faster	No

20.11.1.3   Codify pull-request "Introduce scopes during tracing" #3016

From Ziyu Bao:

Comments Index	Code
1	Introduce scopes, not accepted
2	Try-finally suggested
3	Change from 1 to multiple variables
4	Choices to make strings interned kept
5	Capture module name
6	Context manager for scopes/try-finally
7	Args to `__call__` in module are flattened (instead of `_first_var`); introduced a `torch.jit.scope` context manager
8	Tracing_state in try finally for later proper cleaning-up
9	Setstate, child name

Comments Index	Code
10	Remove if scope.empty in pushscope
11	Remove if the value is not none
12	Context manager activated only when tracing to reduce cost
13	Module name wasn't gotten right, fixed
14	Printing of scope printed only if defined
15	Fixed scopes for the ONNX pass
16	Trying to merge
17	Cannot share common subtries between tries and not needed
18	Appeal to merge, discussion on scope stability
19	Add underscores, others are exposed on .torch namespace
20	Use unique_ptr safely
21	Raise an error if it can't pop
22	Return scope name in scope class
23	Fix numerous small issues
24	Define jit in the module and attach the module to tracing
25	Activate dynamic attributes on TracingStat; manage a stack per trace
27	Build finished
28	Double check: Scope ownership scheme
29	Merged

From Zhao Yin:

Comments Index	Code
1	Usage showed
2	Small advice to use try-finally.
3	Technical advice about function usage.
4	Add name as the attribute to the module.
5	Add context managet to handle the scope.
6	Integration of advice above.
7	Cope with field missing after adding the name.
8	Remove passing an empty string to the decorator.
9	Remove redundant condition check.
10	Remove context when nothing is tracing.
11	Fix $O(n^2)$ to $O(n)$ of loading module tree.
12	Fix scopes for the ONNX pass.
13	Wait for `autograd` PR because it changes JIT infra.
14	This PR get rebased on the new master.
15	Make notes that different `Graph` can't share portions of the trie.
16	Good enough, need to merge.
17	Two concerns: backward tracing with the scope not working and a "scope" is morally a property of the tracer.
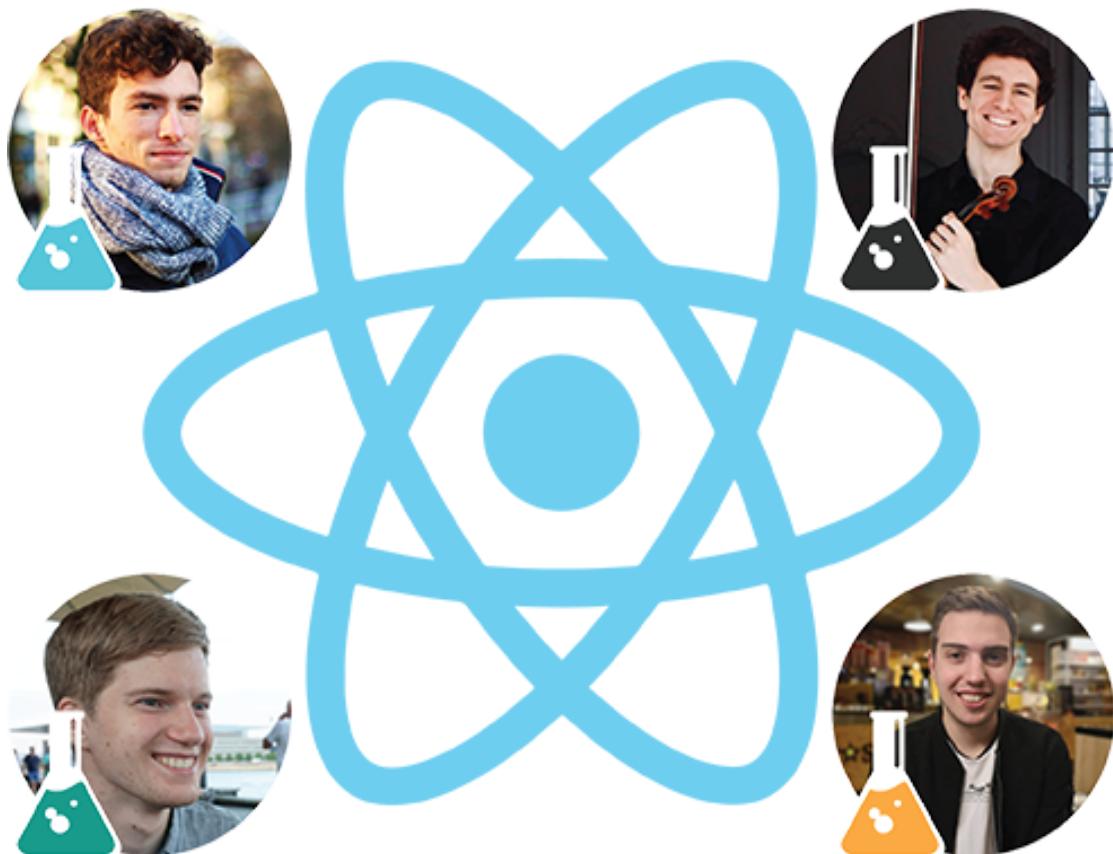
Comments Index	Code
18	Clarifications for the two concerns: making scope inexpensive and a `Graph` node needs to hold information.
19	State that the scopes are stable if name registered in `Module` is stable.
20	Add an underscore and leave others exposed to `torch.`.
21	Change pointer type for the root, scope, and children.
22	Raise error when can't pop.
23	Question (with the answer of not using the flat list of scope) about reversing.
24	Put string representation inside `scope` class.
25	Many small issues fixed.
26	Create the field in tracing state to store extra info.
27	Activat dynamic attributes on TracingState and manage a stack per trace.
28	Build finished.
29	Change the scope ownership scheme.
30	Use vector as children container in scope trie.
31	Avoid segfault in case of memory allocation error.
32	Changes approved. Finally merged.

Chapter 21

React Native

J.S. Abrahams, G. Andreadis, C.C. Boone, F.W. Dekker

Delft University of Technology

21.1 Table of Contents

21.2 Introduction

With the mobile form factor becoming increasingly common for consumer applications, the quality, and coherence of user experience are more important than ever. No longer are users satisfied with services wrapping a web application in a web view container and publishing it as a mobile application. Users expect native, high-quality interfaces which fit well in their host's ecosystem. React Native provides a framework for developing such interfaces, in a cross-platform fashion. By writing JavaScript code with React components, developers can leverage native Android and iOS components and APIs.

In this chapter, we present our view of the React Native framework. First, we take a look at the stakeholders involved in the project and place it in its broader context. We then dive into React Native's architecture and implementation. Following this, we investigate the process that the React Native team employs to integrate changes and evaluate the past and current state of technical debt in the system. Finally, we analyze how developers use React Native and what improvements can be gathered from these patterns.

21.3 Stakeholder Analysis

We conducted a stakeholder analysis to find which parties have an interest in the development of React Native. We categorized these into stakeholder classes as outlined in Rozanski and Woods [1], and added three more classes relevant to this project: competitors, ecosystem enhancers, and integrators.

While Rozanski and Woods distinguish between testers, maintainers, and developers, we found that these are the same people in the case of React Native, and thus combined these classes into the developer class. Finally, since React Native is an open source framework, we have left out the system administrator class.

---

[1]Nick Rozanski and Eoin Woods. Software Systems Architecture: Working with Stakeholders using Viewpoints and Perspectives. Addison-Wesley, 2012.

21.3.1   The Role of Facebook

React Native is the outcome of an internal Facebook hackathon in 2013 [2]. Since 2015, it is available as an open-source framework [3]. Initially, it only had iOS support, but Android support was added after a few months. Facebook had already built two apps using React Native at the time it was released: Facebook Ads Manager and Facebook Groups [4]. Currently, the company also uses React Native for the Facebook app [5].

React Native is thus an important project for Facebook, both as a developer and as an end user. Therefore, Facebook invests in the development of React Native and has a development team working on it [6]. Since they are so involved in the product, Facebook appears in almost all stakeholder classes as a relevant stakeholder.

21.3.2   Stakeholder Classes

21.3.2.1   Acquirers

According to Rozanski and Woods, the group of "acquirers" typically includes senior management [7]. It is easy to conclude (from "The Role of Facebook") that the members of the senior management team at Facebook are the acquirers: They fund the internal development team, and they need the end product to run their business.

21.3.2.2   Assessors

All contributors are required to sign the Contributor License Agreement (CLA) before their PRs are even reviewed. The terms and conditions of this CLA include provisions placing the responsibility for third-party license infringements on the contributor and ensuring the transfer of ownership for accepted contributions.

To assess whether PRs are compliant with the project's standards and expectations, React Native makes use of bots that perform preliminary checks (e.g., CLA compliance, code style), in addition to the automatic testing done by CI services.

If there are no remarks from the bots and the automatic tests pass, the integrators are responsible for further checking of compliance. Legal compliance of PRs is further checked by the core team's PR reviews, CI checks (both on GitHub and Facebook's internal systems), and perhaps some internal checks behind the scenes.

21.3.2.3   Communicators

Communication about React Native happens on various channels, ranging from Twitter to their blog. React Native also provides more in-depth communication on their website (maintained by hramos and charpeni).

---

[2]https://news.softpedia.com/news/facebook-s-react-native-framework-gets-windows-and-tizen-support-502930.shtml

[3]https://medium.com/react-native-development/a-brief-history-of-react-native-aae11f4ca39

[4]https://code.fb.com/android/react-native-bringing-modern-web-techniques-to-mobile/

[5]https://facebook.github.io/react-native/showcase

[6]https://www.reactiflux.com/transcripts/react-native-team/

[7]Nick Rozanski and Eoin Woods. Software Systems Architecture: Working with Stakeholders using Viewpoints and Perspectives. Addison-Wesley, 2012.

The community GitHub organization hosts several communication repositories such as the discussions and proposals repository.

Within the project, we have identified hramos and cpojer as communicators by analyzing metadata files such as `CODEOWNERS` and looking at commits on documents like README.md and CONTRIBUTING.md.

21.3.2.4 Competitors

Competitors are also important stakeholders, influencing the reputation of the field and the innovation and progress within that field. If a competitor introduces a new feature, React Native may want to implement this feature as well, to keep its user base. We list the competitors of React Native in the Power-Interest diagram later in this section.

21.3.2.5 Developers

React Native has well over a thousand different contributors. Some of them stand out in particular: the core team and the most active contributors. We have analyzed the 15 most active contributors in terms of the number of commits [8]. We see five members of the core team in this list, but also seven other Facebook employees. Only 3 of the top-15 are not Facebook employees. The full analysis of top contributors can be found in the appendix.

21.3.2.6 Ecosystem Enhancers

The ecosystem of any framework consists of its ecosystem enhancers, which may be build tools and quality control mechanisms. As such, ecosystem enhancers provide a mutualistic relation with React Native: They could not exist without React Native because they are built especially for the language, but without ecosystem enhancer no-one would want to use React Native.

There are various community-supported projects that bring React Native to other platforms, such as React Native Windows, ReactXP, and React Native macOS, but there are also dedicated toolchains such as Expo. Other relevant ecosystem enhancers are IDEs such as JetBrains WebStorm, which facilitate writing programs in React Native and thereby make React Native more attractive to use. As such, ecosystem enhancers are interested in and have some power over React Native.

21.3.2.7 End-Users

Indirectly, React Native is being used by more than a billion end-users worldwide [9]. These end-users are typically not interested in the technical details that make an app work. However, they care about non-functional properties that are a direct result of these details. If React Native allows the developers to decrease turnaround time on issues and improve cross-platform support, this can benefit the end-users of the apps made with React Native.

---

[8]https://github.com/facebook/react-native/graphs/contributors
[9]https://newsroom.fb.com/company-info/

21.3.2.8   Integrators

Integrators are responsible for deciding whether a PR should be merged, and thereby determine the direction React Native evolves in. The React Native repository assigns specific users as the "code owners" of parts of the codebase and requires that these code owners approve any PR touching this code before it can be merged. React Native currently has five code owners with different responsibilities:

- shergin – The core
- cpojer, hramos – Markdown files and dependency management
- janicduplessis – The animation framework
- mhorowitz – The C++ bridge

21.3.2.8.1   Analysis of Integration Strategies   The daily work of the integrators consists of various (often implicit) decision strategies. Following the research of Gousios et al. [10], we now take a look at how integrators decide when (not) to merge a PR. We start by analyzing the comments of integrators on 25 randomly-sampled accepted and rejected PRs. Since it is often not possible to accurately interpret the strategies that were considered by an integrator in the merging process, we only report on the ones that we could categorize without doubt.

- Code Quality: #24162, #24118, #24066,
- Functionality: #24167, #24063
- CI Conformance: #24070
- Duplication: #24054

We see a similarity between these preliminary results and the results by Gousios, as both show a heavy emphasis on code quality in the review process. Looking at the challenges that integrators face, we identify a lack of time as the primary challenge, likely due to the high volume of incoming PRs. Another challenge revolves around testing, which is not always stringently applied in PRs. This presents integrators with a trade-off between accepting under-tested functionality quickly, or rejecting it and causing delays. Finally, we see a challenge in assessing the functional quality of submitted code. In multiple cases we see integrators referring PRs to other integrators, due to content-related questions.

21.3.2.9   Suppliers

According to Rozanski and Woods [11], suppliers "build and/or supply the hardware, software, or infrastructure on which the system will run." We already mentioned Expo as an essential ecosystem enhancer. However, since it provides the default way of building and running a React Native application during development [12], it is also a significant supplier. In the end, applications built with React Native run on iOS and Android. Changes to those operating systems will also influence the way React Native apps are generated.

---

[10]http://www.gousios.gr/blog/How-do-project-owners-use-pull-requests-on-Github.html

[11]Nick Rozanski and Eoin Woods. Software Systems Architecture: Working with Stakeholders using Viewpoints and Perspectives. Addison-Wesley, 2012.

[12]https://facebook.github.io/react-native/docs/getting-started

21.3.2.10 Support Staff

Although an open source project like this has no dedicated support staff, React Native users can search for help over various channels. If a user has found a bug, they can open an issue on the React Native repository. For questions, users can go to the Reactiflux Discord server, React Native Spectrum chat, StackOverflow forum, or the Expo forums.

21.3.2.11 Users

React Native is used by many companies (to improve consistency between platforms and reduce development time) and individual users (due to familiarity with the React JavaScript framework). Facebook is likely the biggest user [13], though Microsoft also actively uses the framework [14]. We list other users in the Power-Interest diagram later in this section.

21.3.3 Power-Interest Diagram

We visualize the stakeholders in this section based on their power and interest in the project in the figure below:

21.4 Context View

In this section, we place React Native in its greater context. We portray its interaction with the environment, which includes people, systems, and external entities with which it reacts.

As React Native is a framework for cross-platform mobile development, a key requirement is that developers can write platform-agnostic code. Another critical requirement is that the performance of applications written in React Native is comparable to applications written in the languages of those platforms. In general, users of React Native are developers, meaning React Native fits between those developers and the iOS and Android systems, respectively.

Going clockwise, users are the ones that use React Native to build their (mobile) applications (see Stakeholder Analysis). React Native is deployed using the npm package manager [15]. Development tools also make it easier to develop React Native applications. Such tools include Yoga, an open source tool which allows the creation of flexible layouts on multiple platforms [16].

Facebook is the creator and sponsor of the project, which has an MIT license. The developers of the project are described in the Stakeholder Analysis. These developers use several communication channels to communicate with one another and the core team. GitHub provides the infrastructure for open-source development, and it is also where issues are tracked. AppVeyor and CircleCI are used as the automatic continuous integration providers for early detection of problems.

---

[13]https://facebook.github.io/react-native/showcase
[14]https://react-etc.net/entry/microsoft-office-rewrite-to-react-js-nears-completion
[15]https://facebook.github.io/react-native/docs/getting-started
[16]https://facebook.github.io/react-native/docs/more-resources#development-tools
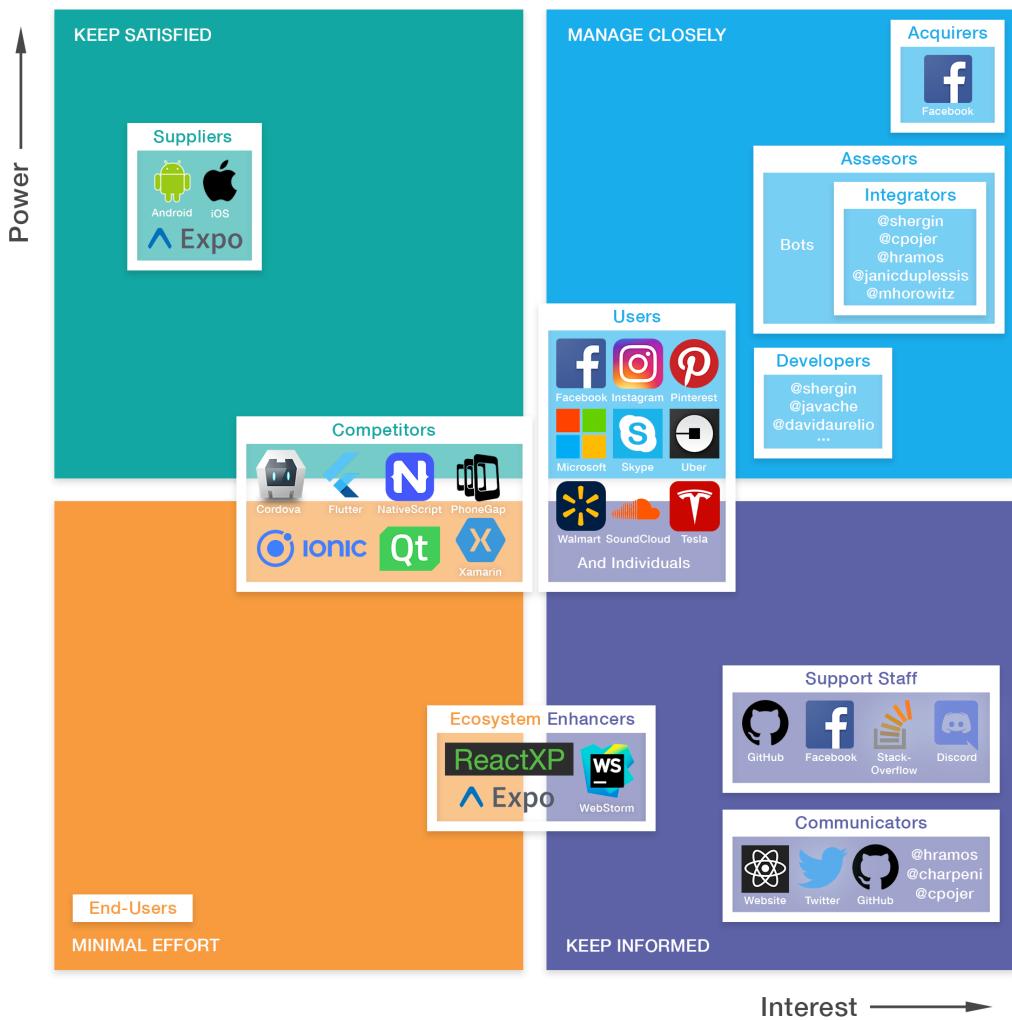
Figure 21.1: Power Interest
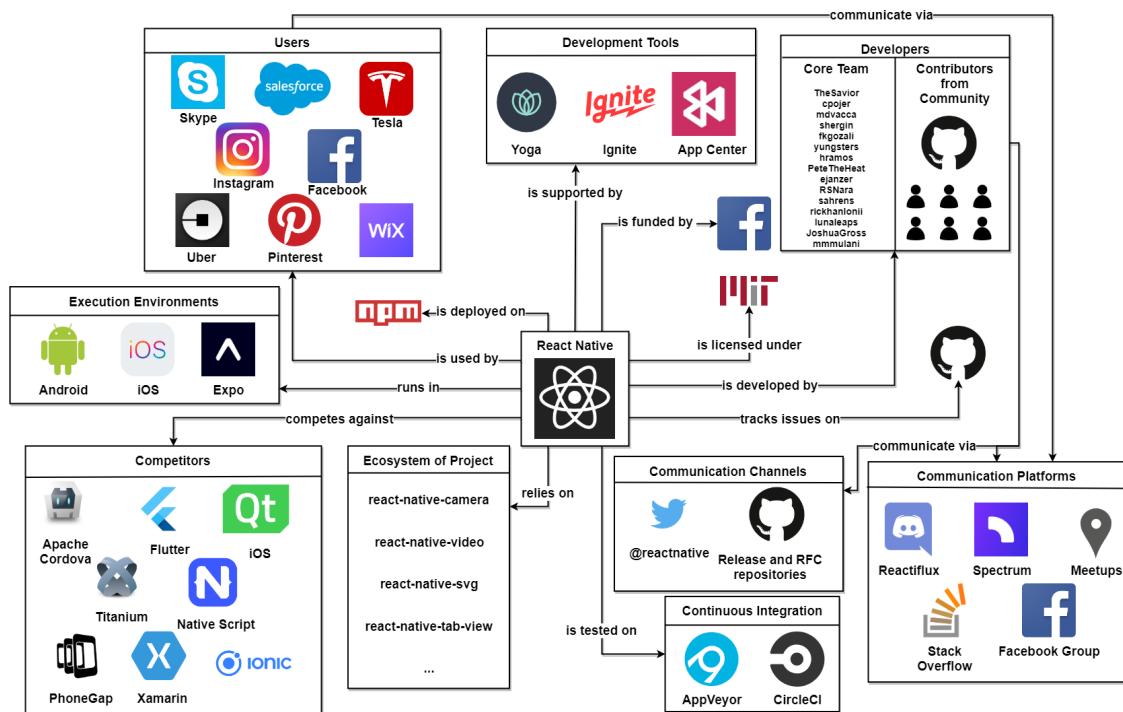
Figure 21.2: Context View

There are many projects within the React Native ecosystem that provide extra functionality. The project also has several competitors, see Stakeholder Analysis. React Native apps run on Android and iOS devices, and for developers also using Expo.

21.5   Development View

For large projects, it is essential to carefully consider the code structure and identify common processes. To achieve a maintainable codebase, it is also important to standardize issue tracking, code style, and testing, to name a few. This section outlines the development view, which aims to provide an overview of these key aspects of the system.

21.5.1   Module Organization

Fundamentally, React Native runs on three threads: the native thread, the shadow thread, and the JavaScript thread [17]. Users interface with the JavaScript thread to modify the state of the application. When the interface needs to be updated, the JavaScript thread asynchronously calls the shadow thread, which maintains a virtual platform-independent DOM called the shadow tree. The shadow thread's reconciler occasionally calculates what has changed since the last update and tells the native thread what should be changed, calculating the desired positions of components using Yoga. Communication between the native and shadow thread happens over the bridge, which (asynchronously) translates JavaScript objects to native objects and back. Finally, the native thread is where the actual UI updates take place by directly interacting with the DOM.
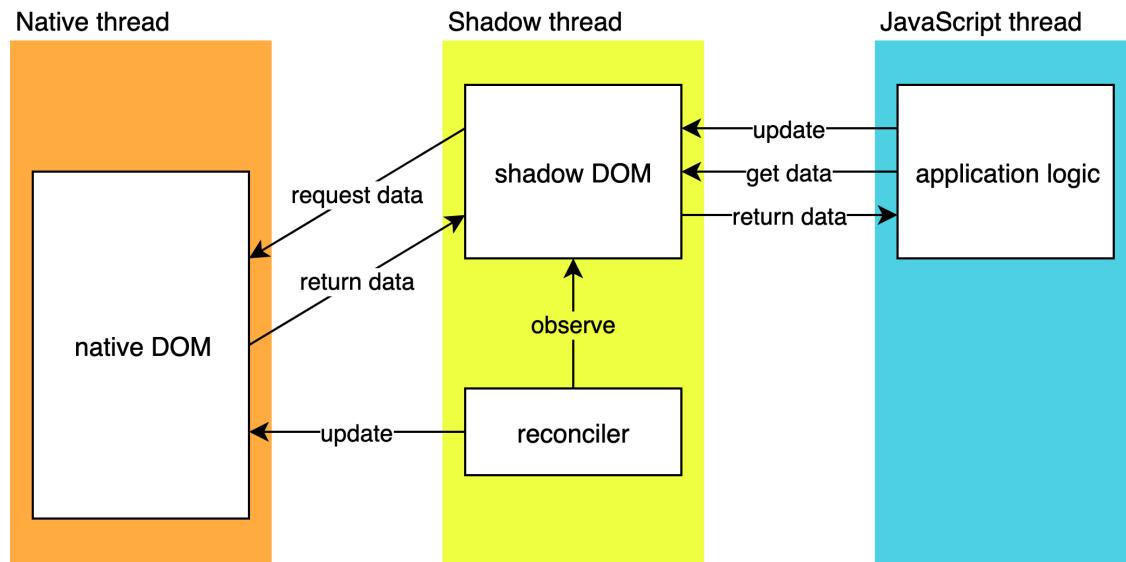
Figure 21.3: Old module organization

---

[17]https://www.qed42.com/blog/react-native-fabric-why-am-i-so-excited

Currently, React Native is in the process of migrating to a new architecture [18]. The new architecture replaces the bridge by the JavaScript Interface (JSI). With the JSI, the interaction between the shadow thread and the native thread changes radically. Rather than translating JSON to and from native objects asynchronously, the JSI will expose the native DOM as a JavaScript API so that the shadow thread can interact with it directly. The benefit of this approach is that the shadow thread will no longer need to maintain a virtual DOM since it can directly change the native DOM [19].
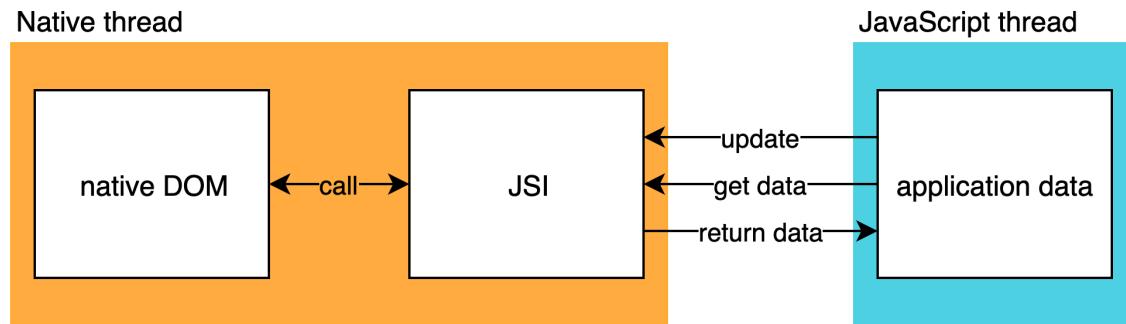
Figure 21.4: New module organization

21.5.2 Common Processing

Large systems, in general, benefit from isolating common processes into separate core modules. A trivial example would be defining a single logger, which allows standardization of logging behavior across the codebase. React Native contains only a few directories which can be classified under common processing. One example is the `ReactCommon` directory, which contains the bridge and runs on the shadow thread. While common to both platforms, it is only loosely classifiable under common processing: It does not so much contain processes which are common to the codebase but is more a link between native and JavaScript code on a conceptual level. `Libraries` contains several JavaScript libraries which interface with Android and iOS libraries, forming a core part of the framework interface. `Libraries` itself is not classifiable under common processing, but it does contain a directory `Utilities`, which embodies a common processing component, as it contains often used functionality such as `logError` and `logInfo`, and a `Dimension` class.

21.5.3 Development Process

The contributing document outlines how developers can contribute to React Native, and includes, amongst others, sections on code style or test plans. The code of conduct highlights what developers should adhere to and what is and is not tolerated.

React Native uses GitHub issue tracking to keep track of feature requests, bugs, and to a certain extent general discussions. With over a dozen new issues per day, issues are tagged automatically by bots.

---

[18]https://facebook.github.io/react-native/blog/2018/06/14/state-of-react-native-2018
[19]http://blog.nparashuram.com/2019/01/react-natives-new-architecture-glossary.html

For every pull request, aside from an explanation as to why it should be merged, developers are asked to write a test plan. It should show that the added functionality works, how errors are handled, and that existing functionality is not tampered with. Test plans often contain screenshots of added functionality.

PRs are not directly merged into the `master` branch. Instead, when accepted, they are closed and integrated into the internal repository of the React Native team. The changes are later synchronized with GitHub.

Standardization of code style helps to keep the codebase clean and easy to read. Prettier is used for formatting of JavaScript code. Lint is used to format code in general, including Java and Objective-C code.

21.5.4 Standardization of Testing

React Native has tests for multiple platforms at several testing levels. These are run automatically for every pushed commit. Various test environments are used because React Native is written in several programming languages.

- For JavaScript, the project uses Jest to execute tests, and Flow as a type checker.
- Java tests are executed using JUnit 4. The integration tests run on a (potentially emulated) Android device. The Flow type checker checks the API exposed by Java code through the JSI.
- Objective-C is tested using integration tests and end-to-end tests. The React Native team has written their own testing tool, RNTester, which executes tests written in JavaScript. In addition to regular integration tests, screenshot tests are used to check that applications are rendered the same as in previous versions. The end-to-end tests additionally require Detox to emulate user input. The Objective-C APIs are also type-checked using Flow.
- There are several end-to-end tests that use most of the environments above.
- Finally, Facebook runs a number of closed-source internal tests.

21.5.5 Codeline Organization

There is no single standard that describes the code structure because the various parts of React Native are written in different programming languages. Instead, the repository consists of several overlapping structures. For this section, we have attempted to extract the different codelines and describe them individually.

21.5.5.1 Android

Android-specific code resides in the `ReactAndroid/` folder. This folder is structured like a standard Android Gradle project: configuration in the root folder, production code in `src/main/`, unit tests in `src/main/test`, and integration tests in `src/main/androidTest`. The `jni/` (Java Native Interface) directories contain C++ code which can be invoked directly from Java code and is linked to the bridge/JSI. The `java/` folder contains the Java source code. This is divided into several packages, such as one for drawing React components and one for creating layouts.

21.5.5.2 Objective-C

Unlike the Java code, the Objective-C code, found in `React/`, has a very flat structure: The `React/` folder immediately contains the modules responsible for drawing and the layout. The integration and end-to-end tests of Objective-C reside directly inside the `RNTester/` directory and are organized by test type.

21.5.5.3 JavaScript
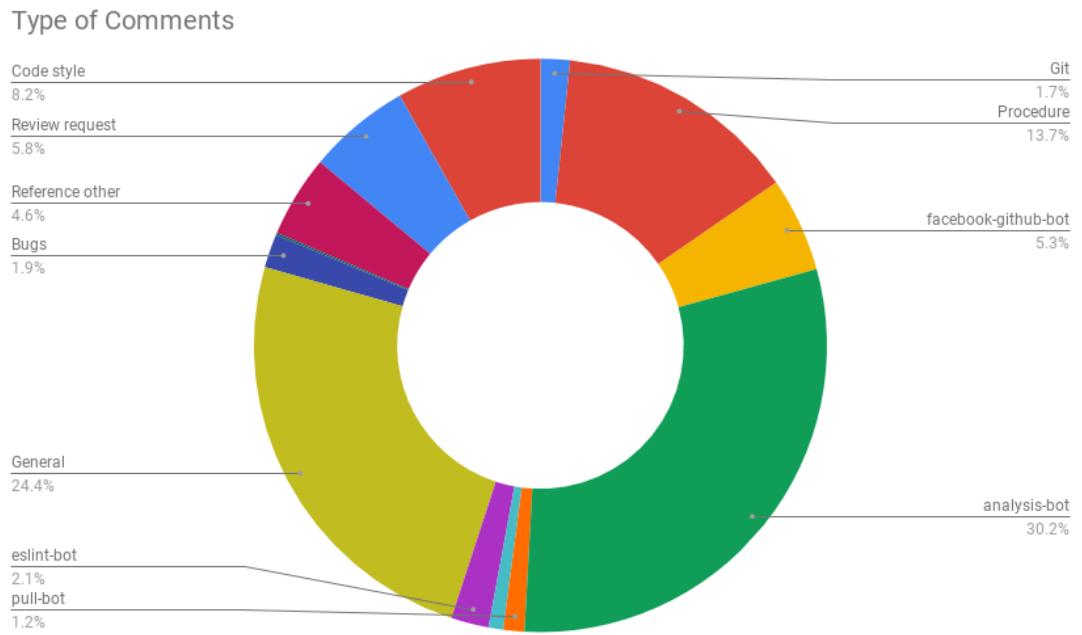
The JavaScript code that the user interacts with can be found in the `Libraries/` folder, where each component then has its own sub-directory. Each component also has tests, which are found in a `__tests__/` sub-directory.
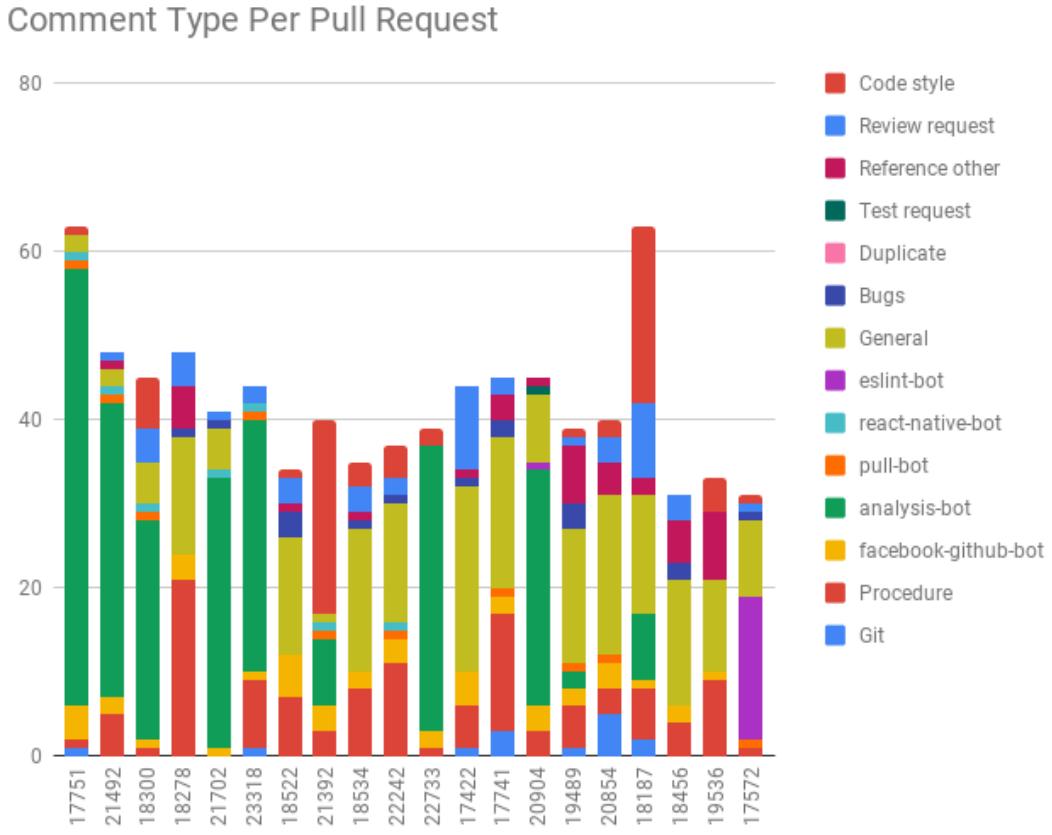
21.6 Pull Request Analysis

We aim to better understand the decision-making process of stakeholders, and the characteristics of how the community operates. To this end, we analyzed 20 PRs from less than one year ago. We start by first tagging each comment on each PR to gain a better understanding of what is happening. For tagging purposes we used the following tags:

Tag	Subcategory
Git	-
Procedure	-
Bot	facebook-github-bot, analysis-bot, pull-bot, react-native-bot, es-lint-bot
Discussion	General, Bugs, Duplicate, Test request, Reference other, Review request, Code style

Three members of our team proceeded to tag each comment and discussed disagreements until we reached a consensus. For each PR, we also noted why it was or was not merged, listed in the appendix. Below we also see a side-by-side comparison of the type of tags for each PR.

Type of Comments

Comment Type Per Pull Request

We observe that user hramos is heavily involved in many PRs and often refers people to other people, as seen in PRs 18522 and 17422, thereby functioning as a bridge.

We also observed that the speed at which PRs are merged seems to rely heavily on how busy the team is. For instance, when analyzing PR 18300, we observe that the team often had to be pinged. In 18278 we even sensed some frustration, due to the problem being urgent and the core team being slow to reply. Furthermore, it took the team over a month to comment on 18534. At the time of this PR, it seemed that the team heavily focused on adding tests to master, indicating an internal agenda.

In general, we see that other developers do comment on PRs, but that the core team has the final say on whether something gets merged or not. PRs are predominantly merged when both the core developers, and occasionally core contributors, have given their stamp of approval.

Based on the above figures, we also made the following observations regarding the comments on PRs:

- More than half of all comments are not related to code.
- There are on average more "General" comments on unmerged PRs, perhaps because these PRs are more controversial, which sometimes leads to them not getting merged.
- There are on average fewer comments by analysis-bots on unmerged PRs. We do not have a comprehensive hypothesis for this.

21.7   Technical Debt

Every software project is touched by it in some form: technical debt. The costs you implicitly introduce when you choose the "easy" solution; the solution that will save you time right now, but will lead to rework being necessary in the future. In this section, we take a look at the technical debt of React Native. How much technical debt does it currently have, and how has this evolved?

21.7.1   Current State

We tried to assess the current state of React Native's technical debt through a combination of automated and manual analysis.

21.7.1.1   Automated Static Analysis

To independently analyze the quality of the codebase, we ran a SonarQube analysis on the following folders: `Libraries`, `React`, `ReactAndroid`, `ReactCommon`. We used the built-in JavaScript and Java scanners, as well as the Backelite/sonar-swift plugin for Objective-C code. We disabled analysis of compiled code artifacts since the codebase was not easily compilable into a central location.

Before we give the full analysis, we first investigate the distribution of code volume over language. A breakdown of line counts over language is given in the figure below:

JavaScript	97k
Java	64k
Objecti...	60k
XML	941
HTML	640
CSS	68
Python	15
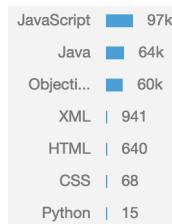
Figure 21.5: Static analysis report

Overall, the main folders of this codebase have more than 224k lines of code.

SonarQube assesses the technical debt in terms of time needed to eliminate code smells. In this project, it estimates the time needed to be 61 days, corresponding to a maintainability rating of A (since the "technical debt ratio" is less than 5%). SonarQube finds 434 bugs, 54 vulnerabilities, 3.5k code smells, and 437 duplicated blocks.

Looking at a random sample of 20 bugs, the majority of the reports seem to be false positives. Most are falsely identified due to the use of the `flow` type checking framework and the `invariant` library (used to formalize consumer contracts in the form of invariant statements). We also found true positives such as cases of dead code and type errors. However, these are located in files marked as debug or development only. The list of false negatives is significantly longer than the list of positives. The list of issues on the repository speaks volumes on the numerous undetected corner cases that were uncovered in real-life use cases.

The majority of the vulnerabilities is classified as minor. We found that minor cases do not immediately impose security risks. SonarQube identifies one major issue relating to the TLS version not being specified, but we identified this as a false positive. Another blocking vulnerability relates to PASSWORD in a name, but this is also a false positive, since it is used in a variable denoting a type of text field.

Next to the false positive code smell reports, some applicable and useful refactoring advice was given in the form of expressions which always return a certain value. To give an example:

```
let shouldStart = true;
// ... code which does not modify shouldStart ...
shouldStart = shouldStart && passesWhitelist;
```

Here, `shouldStart` could be removed from the right-hand side of the assignment expression, as its value does not change. There are many cases similar to this one which could be refactored.

21.7.1.2 Manual Static Analysis

Static analysis can only give part of the picture. We also analyzed a sample of files, manually. While we found general code quality to be high, we identified several design issues, ranging from single-responsibility violations to extensibility concerns. The appendix of this chapter contains the full results of our analysis.

21.7.1.3 Test Coverage

The total test coverage for this project is hard to measure, due to the many languages involved in different parts of the codebase. As JavaScript is the primary language of this codebase, we focus on the report generated by the Jest testing tool here. In total, 8.7% of all branches and 11.9% of all lines are covered. More than half of all components in the `Libraries` folder have no tests at all, while only roughly 20% of all components have more than half of their statements covered.

It is difficult to establish the root cause for this high testing debt. We speculate that a substantial factor is the lack of a coverage step in the CI pipeline; contributors are not confronted with their test coverage. The only thing they need to submit is a "testing plan," which is an informal description of how they tested the added code. Often, this is a description of manual tests. Our advice to the React Native team is to add a stage to the pipeline checking that the difference in test coverage is below a certain threshold.

21.7.1.4 Assessment

In conclusion, we observe that the code quality of this project is high, compared to similar projects. The problem, in our assessment, lies in the testing debt of this project. The test coverage measure on its own is only a metric, but it reveals a deeper issue: A lack of formal tests capturing the intended behavior of components. This means that changes to the code might introduce regressions, in unexpected ways.

21.7.2 Developer Discussions on Technical Debt

As we have shown in our analysis, React Native is no exception from the prevalence of technical debt in large projects. Perhaps more important, however, is how the team talks about technical debt.

Searching for `debt` (verbatim) yields few but telling results. Filtering out accidental mentions, we can see that users are concerned about their technical debt when using the project: In #20508, a user discusses how one would need to take on technical debt if one were to put off upgrading due to a breaking change in a new version. In #6184, a user points out that one should not use functionality about to be deprecated, due to the technical debt involved. Regarding the technical debt of the codebase itself, one team member refers to technical debt in #15232 explaining the trade-off between technical debt and backward compatibility. Finally, we also observe some awareness of the testing debt (e.g., in #23561), although it could be more widely discussed.

Awareness of technical debt is perhaps more visible in implicit documentation within the codebase, as there are 1,466 `TODO` comments and 211 `FIXME` comments in our revision of the project. We conclude that the developers are aware of the technical debt implications of the changes they make. However, we believe that making it a more prominent concern in issue discussions and prioritizing reducing the overall technical debt would help greatly.

21.7.3 Evolution

In this section, we give a brief historical overview of two perspectives on technical debt: static analysis violations and test coverage.

21.7.3.1 Static Analysis

We first look at static analysis. We ran SonarQube on 6 of the past 50 releases, spaced evenly (time-wise). As pointed out in our analysis of the current state, static analysis may yield many false positives. However, it generally gives an indication of technical debt. The plot below visualizes the evolution.

Although not consistent over all metrics, there seems to be a general increase in SonarQube violations over the past two years. Especially the number of bugs seems to increase steadily. The root cause of this is difficult to establish, but it remains an indication of an increasing amount of technical debt.

21.7.3.2 Test Coverage

We now turn to the evolution of test coverage. Since historical data for this was not publicly available, we have written a script which computes the coverage of the JavaScript part of the codebase for the last 50 releases. In the figure below, the results of this analysis are plotted against their release date. Releases where coverage could not be computed are excluded.

Because React Native follows a release model where releases are placed in a dedicated branch and snapshot tests are generated on that branch, coverage of releases is higher than the coverage of day-to-day commits.

Overall, we see a decline in line coverage over time, starting from over 50% and dropping below 40%. Branch coverage has stayed relatively stable. To analyze this further, we plot the covered and total lines/branches, individually:

It becomes apparent that the increase in lines is not met as well in coverage as the increase in branches. This likely can be traced back to long linear blocks of code, which do not significantly contribute to branch complexity but do impact line coverage.
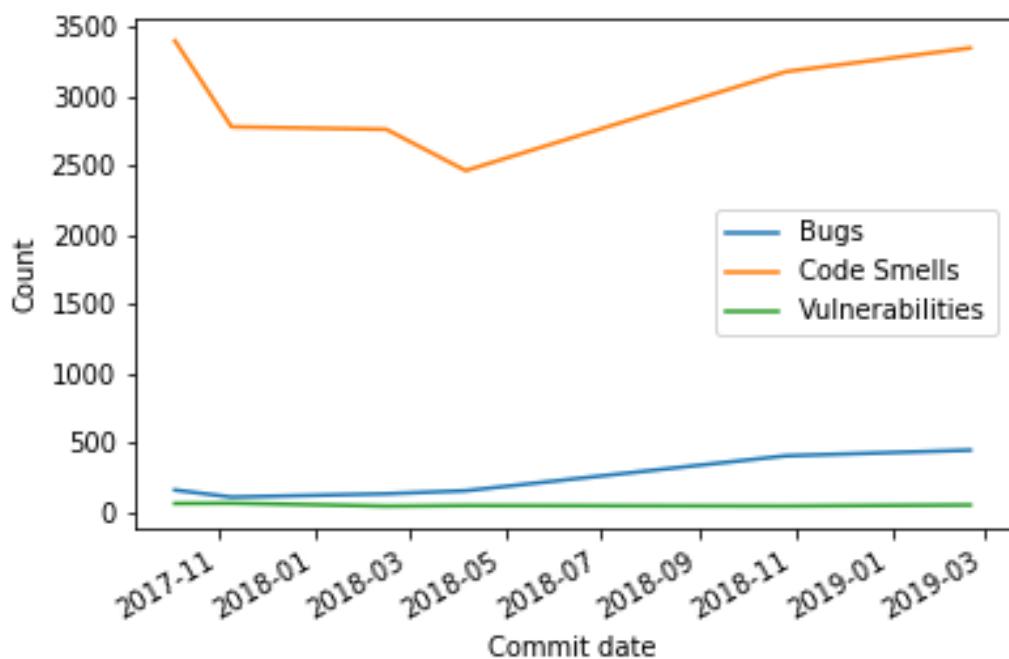
Figure 21.6: Sonar evolution
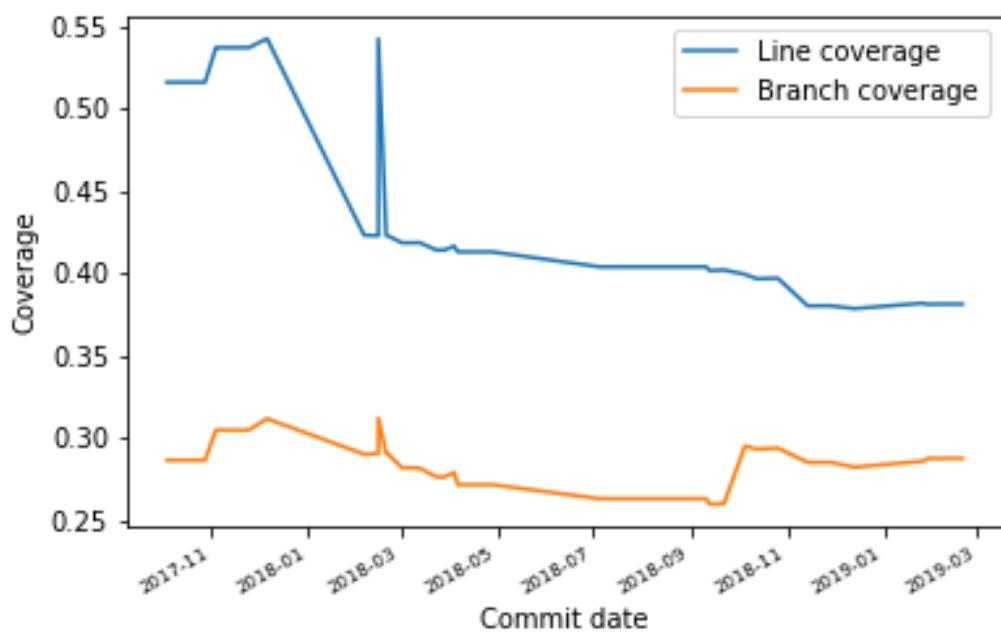
Figure 21.7: Coverage evolution
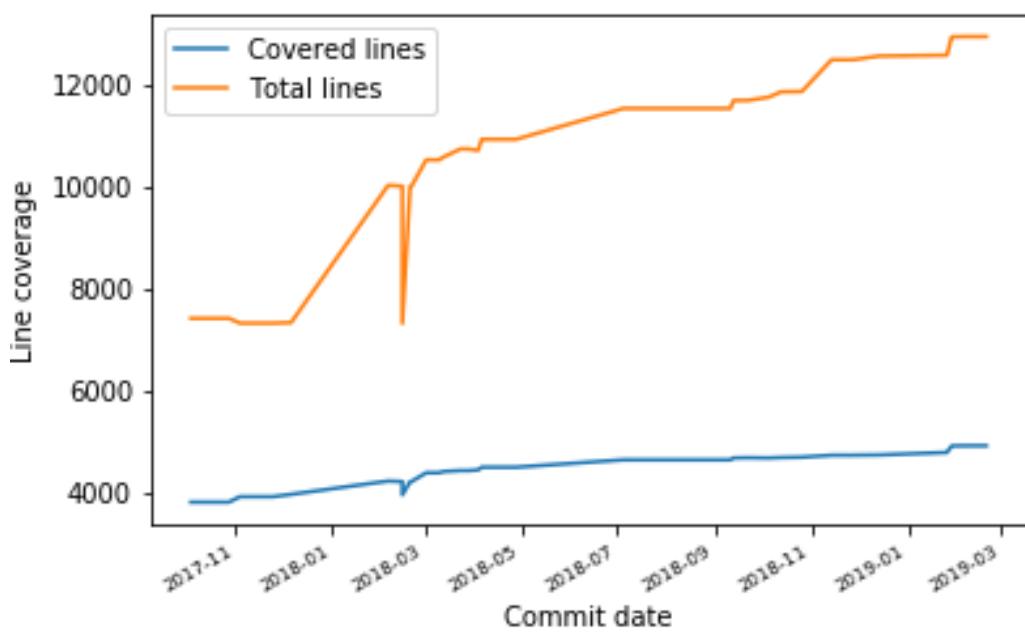
Figure 21.8: Line coverage evolution
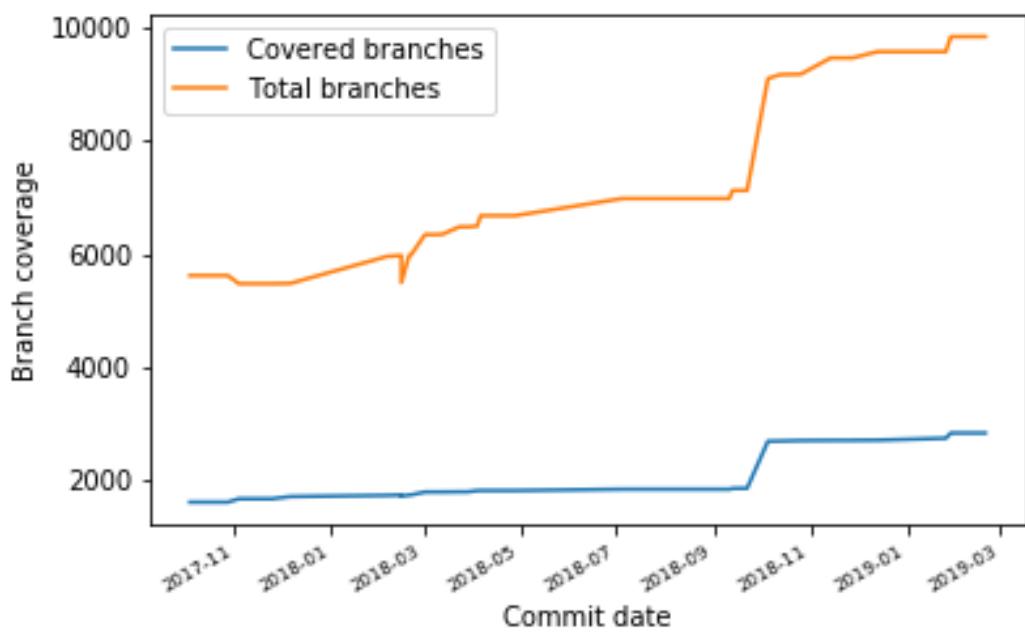
Figure 21.9: Branch coverage evolution

21.7.4   Contributions to Decrease Technical Debt

After we identified the testing debt as the most pressing issue in this project, we made contributions in the form of tests to parts that were sparsely covered before. In the appendix, we list the PRs we opened in order to reduce the testing debt.

To see the decrease in testing debt by adding the new unit tests, we now compare the test coverage before and after our contributions. The appendix contains a description of the process and a detailed breakdown per component.

All Files	Before	After
Branches	8.70% *2565/29485*	8.90% *2625/29485*
Lines	11.87% *4113/34639*	12.15% *4207/34639*

In hindsight, writing the tests took us quite some effort. Although the coverage increased a bit, only 12.1% of the lines in the codebase are now covered. We conclude that there is still significant testing debt present in the codebase. The actual cost of this debt remains elusive: Which bugs that could have been detected by tests will go by unnoticed? There is likely no clear answer to this, but we prefer staying on the safe side by covering more of the codebase with tests.

21.8   Architecture of Apps Written in React Native

React Native is a framework for writing apps. Each of these apps, written by different developers, has a unique architecture. We will evaluate if React Native provides enough helpful means for developers to architect their apps well.

21.8.1   Evaluated Apps

We looked into the following open-source apps.

21.8.1.1   QR Code Reader and Generator

QR Code Reader and Generator is an app which can be used to scan and generate QR codes. The project was started using the Pepperoni App Starter Kit. QR Code uses version 0.49.3 of React Native, released in 2017. The reason for this does not seem to be a difficulty of upgrading, but rather inactive development.

21.8.1.2   F8

The F8 app by Facebook is the official app for Facebook's conference. The app helps attendees schedule their visit and navigate through the conference. Facebook has written a tutorial on how the F8 app was made. As such, the app's code is organized very simplistically, and its architecture is well-explained in the tutorial.

It does not use the latest version of React Native, but that is because the project has not been updated in a while. It uses one of the latest versions that were available at its last update.

21.8.1.3 Mattermost

The Mattermost Mobile app is the mobile version of the Mattermost open-source application. It is an application for team communication.

Mattermost uses version 0.58.5 of React Native. This version was released two months ago at the time of writing. Looking at the history of version updates, we see that the React Native dependency gets updated regularly. From this, we can conclude that it is easy to update.

21.8.2 React Native Enhancers

QR Code and Mattermost use Redux for state management within the app. By default, React and React Native do not provide a way to manage state across an application. F8 initially used Redux, but later transitioned to Relay, by Facebook.

QR Code, F8, and Mattermost each use many packages with small functionalities, such as react-native-sound, react-native-open-maps, and react-native-contacts. These packages all provide very little extra behavior which is relatively common. React Native app developers will need a lot of those packages, which can make the development process harder for beginners and reduces developer convenience. However, including the packages in the core of React Native would make the core less lean. Therefore, there seems to be a trade-off between developer convenience and clean architecture.

21.8.3 App's structure (vs. React Native)

QR Code and F8 do not have any (handwritten) native code. Therefore the codebase solely exists out of JavaScript files.

Conversely, Mattermost does contain some Java and Objective-C, even if the majority is still written in JavaScript. A notable example where they modify the behavior of React Native using native Java code is by using the `AsyncStorageHelper` to access React Native's asynchronous database synchronously for Android.

React Native is not opinionated in terms of file structuring. It only has an `App.js` file in the root and leaves the rest up to the developer. For developer convenience this is not optimal since it does not provide them with a clearly defined way of working. It might be a good idea to provide a suggested "architecture" in the docs, or even create a new (extra) template which takes a more opinionated view, which can also prove valuable for beginners.

There exist resources with suggestions for the architecture of a React Native app. An example is the Pepperoni App Starter Kit, which QR Code uses.

21.8.3.1 Pepperoni

The JavaScript files are structured using the approach suggested in the Pepperoni App Starter Kit. The code is organized into two main directories: `components` and `modules`. `components` contains "dumb" React Native JSX components that can be used by the (more involved) `modules`. `modules` is where all code that "that modifies application state or reads it from the store" should go [20].

Modules are represented as a directory and form a part of the domain. A module contains three logical parts: State, View(s) and Container(s). The State holds data and operations on this data. The Views are the presentation layer and should use State, but not be stateful themselves. It may use components from the `components` directory. The Containers are responsible for connecting one or more Views to the Redux store.

As an example, the following modules are present in QR Code:
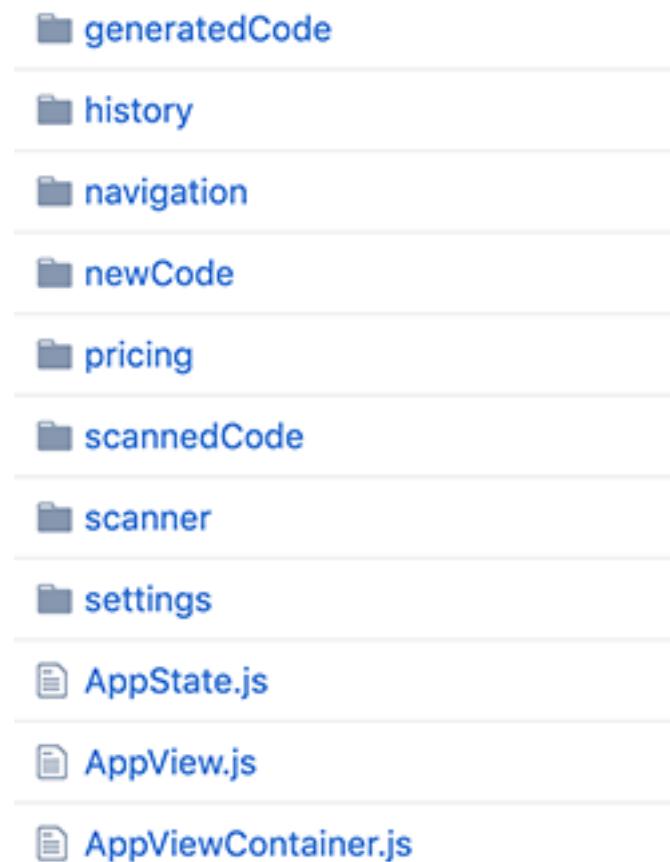
Figure 21.10: QR Code structure

21.9   Testing and Software Quality

All apps have unit tests written using Jest. Both F8 and Mattermost have very few tests. While the React Native documentation does not give any hints on how to test an application, Jest does have such a guide. The integration between Jest and React Native seems to be good.

QR code also has relatively simple end-to-end tests, using Detox. Conversely, the end-to-end tests for Android of React Native itself are written using Appium (which we improved in one of our contributions), which results in complex-looking tests. Detox is written with React Native in mind, resulting in cleaner tests.

It would be a good idea to include a section on testing a React Native app in the documentation, which might help developers in writing their first test or in writing more tests more easily.

21.10   Conclusion

React Native is an important framework for mobile apps, used by many prominent services. Both the development process and codebase are generally of high quality, although we identified significant testing debt. We have made several code contributions to improve the technical debt in the system. Finally, we observe that apps using the framework are highly diverse in structure and believe that a more opinionated guideline could help developers.

21.11   Appendices

21.11.1   Top Contributor Analysis

From the core team we see a lot of contributions from shergin (ranked 1st), sahrens (5), hramos (7), mdvacca (11), and TheSavior (15).

Although the core team consists of Facebook employees, it is interesting to see that other Facebook employees also actively contribute to React Native. In fact, the top 15 most active contributors consists mostly of Facebook employees: javache (2), davidaurelio (3), mkonicek (4, former Facebook employee, currently inactive), vjeux (8, currently inactive), jeanlauliac (10, currently inactive), frantic (12, currently inactive), and bestander (14, currently inactive). Some of these employees may be former core team members.

Finally, we have identified a few non-Facebook employees in the top contributors: nicklockwood (6), tadeuzagallo (9), and janicduplessis (13).

21.11.2   Merged PRs

Id	Reason for getting merged
17751	Simply added an error message, merged due to uncontroversial nature.

Id	Reason for getting merged
21492	Small discussion, author implemented suggested fix.
18300	Most comments were about comment style, eventually merged. **note:** A lot of pining, implying that the core team and/or maintainers was overloaded at the time.
18278	Problem was urgent. **note:** A lot of frustration sensed, due to slow replies and asking for information updates often.
21702	Dominated by bots. Short discussion followed by merge.
23318	Maintainer was quick to point out that a lot of work had to be done. Author implemented changes and eventually large subset was merged.
18522	Interesting technical discussion lead by hramos. Stack trace of internal fb test was shared, before fix was implemented and merged.
21392	Mostly code quality issues, uncontroversial and quickly merged.
18534	At the time this PR was merged, it seems that the internal team was heavily focused on getting tests to work on the master. Only after a month was the PR revisited, and changes from an internal review were applied. After this, it was merged. This reveals the team also has an internal agenda which determines if something gets merged or not.
22242	Supportive discussion, where the author quickly implemented suggestions from the original author before it got merged.

21.11.3 Unmerged PRs

Id	Reason for not getting merged
22733	There was no activity on the PR for over a month.
17422	A long discussion ensued, and eventually it was decided it was better to start a new PR as suggested by hramos.
17741	The functionality implemented by this PR was added to master in the meantime.
20904	Targeted functionality which was moved to a separate repository.
19489	Rebase went wrong so a new PR was opened.
20854	A large subset of this PR was merged during the discussion.
18187	Author did not have time anymore to work on this PR and no one volunteered. However as of writing this PR has been reopened.
18456	Targeted bug was fixed elsewhere.
19536	Functionality of PR was implemented by other PR's in the meantime.
17572	Closed in favour of moving fix to separate repository.

21.11.4 Manual Static Technical Debt Analyses

We analysed a sample of 4 source code files. We present our findings below.

Modal.js: A component used to display modal dialogs. This component has a clear single responsibility and its public interface is well documented. In fact, every property that can be passed to it is individually annotated. The `render()` method might be split into smaller submethods: There are 7 branching points in

this method, which is on the high side. However, this is not as straightforward as it might seem, since the computed values in those branches are needed in the rendering part in the final JSX part of the method.

Geolocation.js: A component used to get the geolocation of the host device. This component also has a clear single responsibility. Two functions (`clearWatch` and `stopObserving`) did not have public documentation. Although the reader might be able to deduce what they do, adding documentation here would help clarify this proactively. We added this documentation in a pull request. We also found an obsolete static analysis suppression comment.

ShakeDetector.java: A hardware interface that detects shakes using the accelerometer. The class has two public methods to start and stop shake detection, and two that are used by the `SensorManager` to indicate that the sensor's state has changed. While the public interface is easy to understand and well-documented, the design may not be optimal. Firstly, the class seems to violate the single responsibility principle because it has two responsibilities: Translating accelerometer measurements to a usable format, and deciding whether the measurements correspond to a shake. This could be resolved by adding an interface in between the sensor manager and the detector. Secondly, the shake detector supports exactly one listener, but it may be worthwhile to make it support multiple listeners.

InterpolationAnimatedNode.java: A drawable node of which the location can be interpolated in between frames. This class suffers from several design issues which make it hard to comprehend. Firstly, the available interpolation methods are hardcoded as a series of strings. A first, naive solution would be to use an enum that lists the methods. This would resolve the issue of not being able to check whether all users of the interpolation method support a newly added method. However, this solution does not allow for adding new methods, and thus violates the open/closed principle. A second problem is that ranges are described as arrays of values, but its bounds are never checked and some functionality is written in helper methods in the node class. This class would become easier to comprehend and safer to use if ranges were described as objects which perform validation internally.

21.11.5   Contributions to Decrease Technical Debt

- Adding tests for utilities: #23903 and #23989.
- Adding tests for the geolocation module: #23987.
- Fixing existing end-to-end tests for Android: #23958.

21.11.5.1   Coverage Improvement Breakdowns per File

We created this comparison by starting at the last commit before our first contribution, `581711c`. We then ran `yarn jest --coverage` to compute the test coverage. To add our contributions, we cherry-picked the relevant (squashed) commits (`47e0615`, `f541c34`, and `6047d42`) and computed the coverage again.

`Libraries/Geolocation`	Before	After (+9 tests)
Branches	0% *0/40*	77.50% *31/40*
Lines	0% *0/49*	91.84% *45/49*

Libraries/Utilities	Before	After (+35 tests)
Branches	34.77% *153/440*	41.36% *182/440*
Lines	48.3% *398/824*	54.25% *447/824*

21.11.6 Contributions

- https://github.com/react-native-community/discussions-and-proposals/pull/103
- https://github.com/facebook/buck/pull/2200
- https://github.com/facebook/react-native-website/pull/805
- https://github.com/facebook/react-native/pull/23804
- https://github.com/facebook/react-native/pull/23903
- https://github.com/facebook/react-native/pull/23958
- https://github.com/facebook/react-native/pull/23987
- https://github.com/facebook/react-native/pull/23989

Chapter 22

SciPy

By Noor ul Sehr Zia, Sharanya Suresha Konandur, Shikhar Dev and Srinath Sudharsan

22.1 Abstract

SciPy is a popular open source software which is widely used for scientific and numerical computing in Python. SciPy is maintained and developed by the core developers on Github as well as external contributors. In this chapter, we study the software architecture of SciPy, precisely its stakeholders, context, deployment, and evolution. Finally, we elaborate upon an analysis of its architecture development and its technical debt. In conclusion, we observe that SciPy is very well engineered project with minor technical holes or debt.

22.2 Table of Content

- Stakeholders View
- Context View
- Development View
- Deployment view
- Technical debt
- Evolutionary Perspective
- Conclusion
- References

22.3   Introduction

SciPy is an open source software library that abstracts and implements key functionalities for scientific computing, mathematics and engineering. It is written in Python and is supported by a large community of developers, researchers and some of the world famous companies including Github, Intel and Travis CI. SciPy has been under active development since 2001 and is now used in a wide range of applications by companies across the world. Some of the prominent modules implemented in SciPy include linear algebra, integration, interpolation, Fast Fourier Transforms, and signal and image processing. In this chapter, we derive views and perspectives from Rozanski & Woods book and perform a thorough analysis of the library. Given the global scale of impact, SciPy provides for a very interesting study in many dimensions. The goal of this article is to provide an overview of key architectural elements of SciPy and to possibly attract more contributors for open source development.

22.4   Stakeholder's Analysis

"A stake holder in a software architecture is a person, group, or an entity with an interest in or concerns about the realization of the architecture." - Software Systems Architecture. Apart from just the users of a software system, there are many different individuals or groups of no individuals who have a direct impact on the system. From conceptualization, design, documentation, implementation, maintenance, to even developing business cases around the software system, development life-cycle of a software involves many different stake holders working in harmony to make a valuable and sustainable system. This section describes the different types of stake holders we have identified for the project, and attempts to identify their contributions to the project. Most of the information below have been sourced from the SciPy github page and the main SciPy website. Minor points have been inspired from numerous blogs, wikipedia and google search.

Type	Brief Description	Owner
Acquirers	Acquirers are stakeholders who oversee the long term and short term strategy of the project. This also includes parties who partly / fully sponsor the project.	A large part of strategic positioning was done by the original creators of SciPy, Travis Oliphant, Pearu Peterson, and Eric Jones. Apart from them, NumFOCUS provides financial support to the project, garnering some influence in the overall strategy.

Type	Brief Description	Owner
Assessors	Assessors oversee the system's compliance to standards and legal regulations.	Python Software Foundation (PSF)
Communicators	Communicators ensure crisp and unambiguous communication of relevant details of the project to appropriate stakeholders.	The SciPy Code of Conduct committee polices all public spaces managed by the SciPy project, and ensures an abuse free productive communication environment. All of the technical documentation and inquiry response is handled by the active contributors active participants in the project.
Developers	Developers are responsible for creating and deploying software solutions that comply with all the expectations.	SciPy originated with code contributions by Travis Oliphant, Pearu Peterson, and Eric Jones in 2001. Since then, significant contributions have been made by over 200 contributors.
Maintainers	Maintainers manage the evolution of the system once it is deployed and operational.	All of the pull requests for SciPy are reviewed and integrated by the community of developers and users of SciPy. All contributors who participate in the PR discussion can be considered Maintainers. In rare cases that agreement cannot be reached, maintainers of the individual module in question decide on the outcome.
Suppliers	Suppliers source the hardware, software, or infrastructure on which the system will run.	Some of the organisations that have funded parts of Scipy operations are: Enthought: scipy.org and mailing lists hosting, holding the SciPy trademark. Enthoughts also sponsors SciPy conferences in the United states and across the world. () *Github: code hosting and development workflow platform* () Travis CI: continuous integration service; SourceForge: hosting released sources and installers; Intel: Intel MKL licenses; NumFOCUS: hosted Mac Mini build machine
Support Staff	Support staff provide support to users for the product or system.	Most open questions and queries are catered to, by the community of users and contributors of SciPy on different channels, including Stackoverflow and Github. Commercial support is offered for SciPy by a number of companies, including Anaconda, Enthought and Quansight.
System Administrators	System administrators ensure smooth operations of the system once it has been deployed.	Since SciPy is offered as a library, no system critical assurance is for the operation. Services such as Hosting, Mailing list, etc are fulfilled by companies listed in Suppliers.

Type	Brief Description	Owner
Testers	Testers systematically test the system to ensure that it is suitable for deployment and use.	All code contributors are expected to perform unit tests on their code before submitting a Pull Request. There is no separate team of testers who test the system.
Users	Users are individuals and organisations who use Scipy for their projects.	SciPy is primarily designed for Mathematics, Science and Engineering. SciPy caters to a wide range of users, including researchers, students, data scientists, as software engineers. The library is used in academic, as well as business setting.

In addition to the above discussed categories which were based on Software Systems Architecture, following categories stakeholders have also been accounted for.

Type	Brief Description	Owner
Event Organiser	Event organisers organise regular outreach and awareness programs.	Conferences are organised by SciPy (in the United States), EuroSciPy (in Europe) and SciPy.in (in India)
Competitors	Competitors include other projects that provide similar offerings.	Some of the most prominent competitors of SciPy are : Matlab; R; Stata
Dependencies	These are entities on which Scipy is reliant upon for services.	Python is the base language SciPy offers its services in; C and Fortran are used for several key components in the SciPy Library; NumPy provides fast and convenient N-dimensional array manipulation capabilities for SciPy.

22.5 Power - Interest Analysis
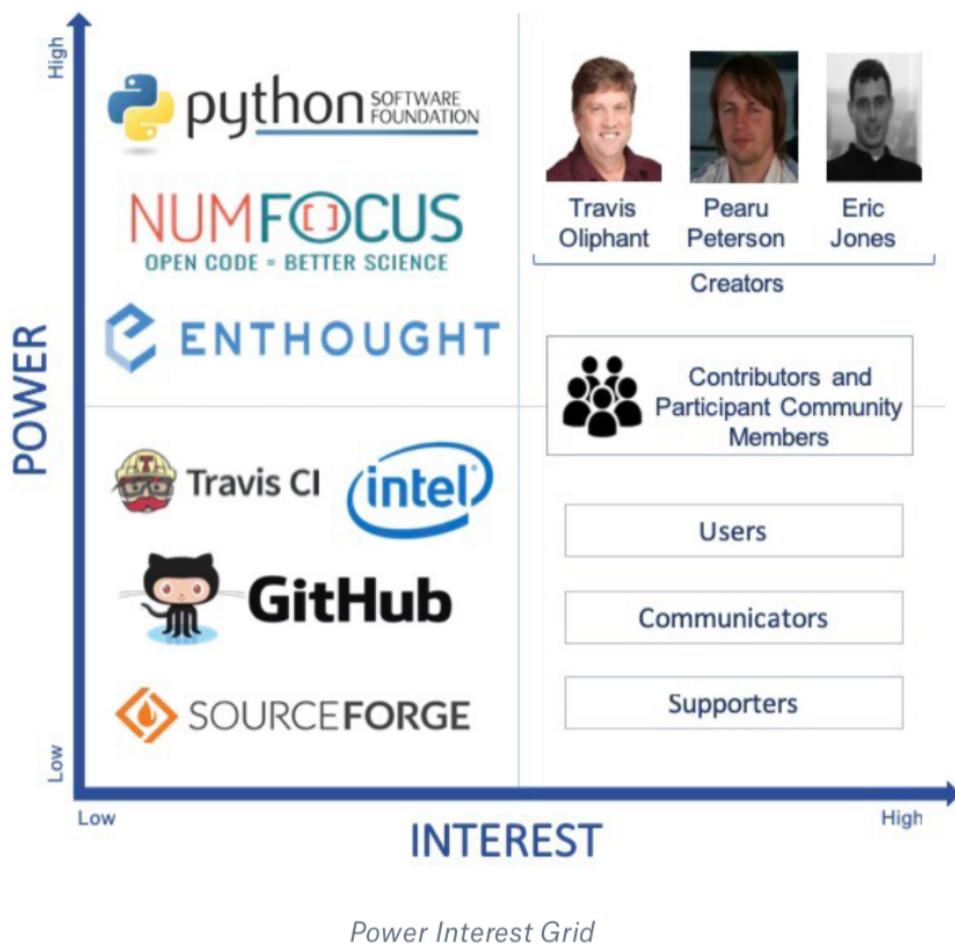
Figure: Power Interest Grid

The power vs interest grid correlates the influence and the interests of different stakeholders involved in the project. The relation can be studied 4 different categories.

- **Low Power and Low Interest** : While this group of stakeholders are invested in the success of SciPy, they do not hold a strong influence on the project, and neither do they have a strong interest in the development activities of SciPy. As shown in the Stakeholders Analysis, Github, Intel, Travis CI and Sourceforge have all contributed services to ensure smooth operations of SciPy - but they do not have any contribution on development activities. [SciPy Donations]

- **Low Power and High Interest** : Users, Communicators and Supporters are stakeholders who actively follow developments of SciPy, but do not have any direct influence on the software project itself. These functionalities are also supported by the open community of par [Contributing to SciPy]

- **High Power and Low Interest** : NumFocus, Enthought and the Python Software Foundation (PSF)

provide vital contributions to the project, but do not derive value from the contributions. NumFocus is one of the primary sources of funds for SciPy. Eric Jones, the co-founder of Enthought was one of the co-creators of SciPy, and had significant contributions to the project. The Python Software Foundation continually develops and improves Python - which has a direct influence on SciPy.

- **High Power and High Interest** : The creators of the library - Travis Oliphant, Pearu Peterson, and Eric Jones hold the largest influence on development activities of SciPy. However, beyond the creators, the library is largely maintained and further developed by a community of contributors and participants of the review process. [Contributing to SciPy]

22.6   Integrators

In analyzing pull requests, we have identified **Ralf Gommers**, **Pauli Virtanen** and **Matthew Brett** as some of the most active contributors and integrators on the the project. They have individually closed numerous pull requests and are very active in discussions under pull requests. While we have not been identified exact areas where we could contribute to SciPy library, it would be very interesting to reach out to them and to get their take on directions they think are important for SciPy.

22.7   Context View

In this section we explore the context view which describes the relationship and interactions between SciPy and other related entities. We visualized and categorized the external entities by their involvement in SciPy. The following image shows the interactions of these entities.

22.7.1   System Scope and Responsibilities

The system scope defines the main responsibilities that SciPy provides, which consists of providing a open and free *Python* library for the development and utilization of scientific and technical computing. Scipy has many responsibility that has distinct entities contributing to the SciPy. The main contributions are:

- Scipy is scientific computing software platform in python which incorporate The SciPy *ecosystem.*
- Scipy also holds exclusive conferences for numerical and scientific analysis in python such as SciPy, cipy.in, EuroSciPy etc.
- Scipy builds on NumPy array which is a numerical stack which provides many user-friendly tools for numerical data analysis and optimizations.
- Scipy is a cross platform software which works on multiple operating systems such as linux, OSX and windows.
- The SciPy library consists of numerical algorithms and software toolbox for signal processing, special functions, FFT, etc.
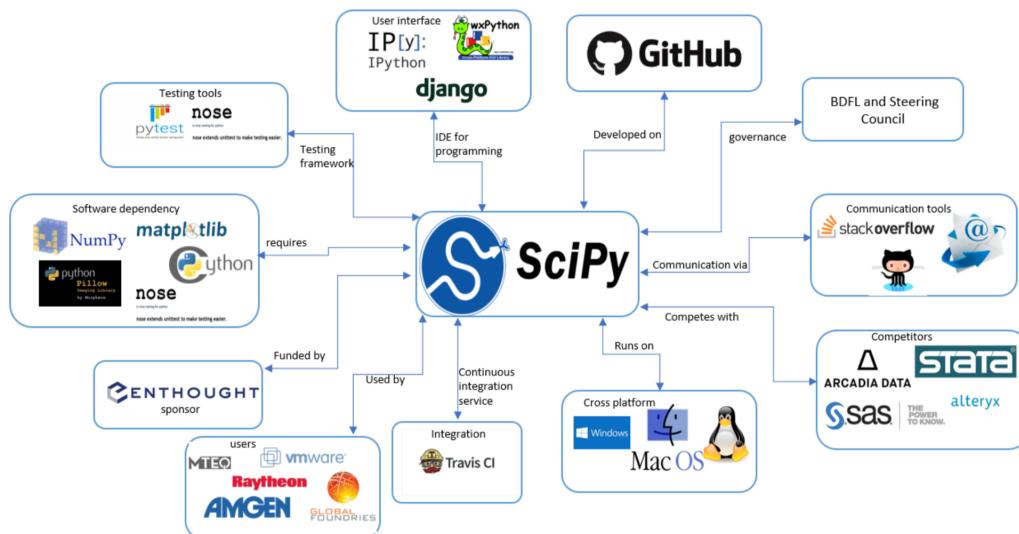
22.7.2 External entities

In this subsection we will explore an how SciPy handles its dependencies. SciPy has several external entities which are listed below, who work as a single community to make the environment for it.

- Developing language: Python
- Sponsored by: Enthought, Inc
- IDE for programming: IPython, wxPython, Traits, Django,etc.
- Communication tools: Stack overflow, Mailing-List, GitHub
- Quality assurance: Nose and Pytest provides framework for python testing.
- Continuous integration service: Travis CI
- License support: BSD license
- SciPy development: Github
- Governance: Pauli Virtanen is the Benevolent Dictator for Life (BDFL) and steering council.
- Competitors: Stata, Alteryx, Base SAS, Arcadia Data.
- Dependency: Numpy, nose (test suite run), asv (benchmarks run), matplotlib (plotting), Pillow (saving images), mpmath (special tests), LaTeX (pdf docs), Cython(development versions)
- Users: MTEQ, Raytheon, Amgen, VMware, Jefferson Frank, global foundries, students, researchers, etc.

22.7.3 Context view diagram

The below diagram summaries SciPy and its entities which devises the environment. It contains the key stakeholders as mentioned in the stakeholder section and the external entities.

Context View Diagram

Figure: Context View Diagram

22.7.4   Pull Request Analysis

We analyzed 20 pull requests from SciPy github repository. The criteria used for selecting the pull requests is:

- Accepted: We sorted the merged and closed pull requests according to their number of comments and took the ones with most comments/discussions.
- Rejected: For rejected, we took pull requests that were closed by the members of SciPy or those that weren't merged in any child pull request. To add diversity, we chose pull requests that were from different labels, like "won't fix", "defect", "maintenance" and "documentation".

22.7.5   Reviewing guidelines

Everyone can review the pull requests, however the members of SciPy are people who perform the final merge process. Moreover, SciPy HACKING.rst provides detailed guidelines about the reviewing process. These guidelines include:

- The change should be discussed in detail and all changes that will follow in existing behavior should be reported.
- The change should be logically and scientifically correct.
- The behavior should be clear under all conditions and all unexpected behavior and exceptions should be reported
- The code should meet the quality, test and documentation expectations outlined in CONTRIBUTING.rst.

22.7.6   Decision making

We noticed that most pull requests that were rejected or closed were due to one of the following reasons:

- The cost of making the change did not exceed the benefits
- There were incorrect assumptions in the logic
- The implemented code did not address all the conditions
- The pull request was not updated in a long time and another fix was already implemented
- The changes proposed were not backward compatible

Through the accepted ones we noticed that some of the reasons a pull request was accepted were:

- It introduced a new method and that method passed all tests
- It optimized an existing routine
- Provided a better alternative to an existing method
- Improved the performance of an algorithm

We have provided a detailed analysis of pull requests in **Appendix A** of our report.

As an attempt to codify the pull requests, we analyzed each comment and assigned a tag to it. To stay consistent in our analysis, we used a common tagging terminology in our codification process. The details of the tags we used and their distribution is shown in **Appendix B.** We have attached the csv files with tags per comment in PR_codified folder in our gitlab repository.
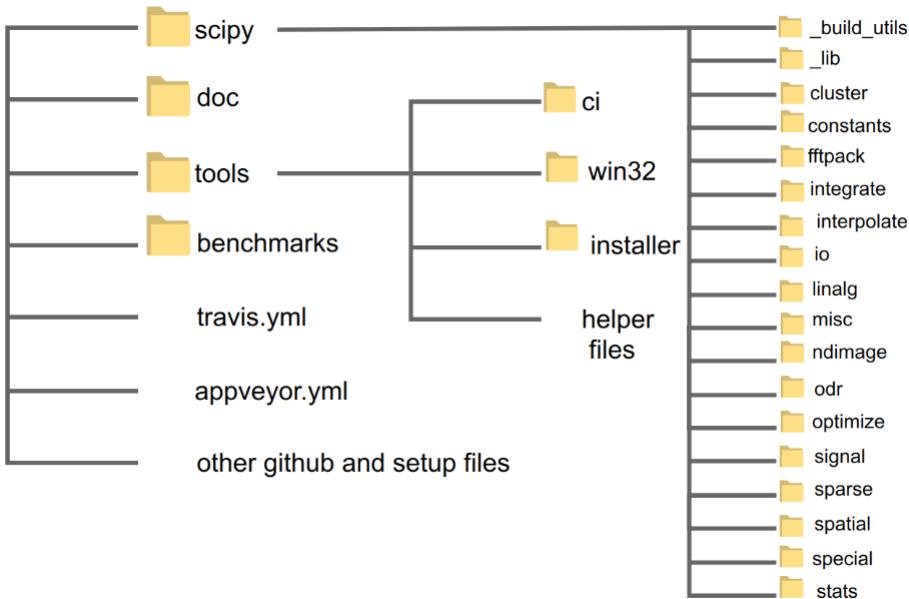
22.8 Development view

In this section, we delve into the code structure and analyse the architecture of SciPy. The primary objective of this section is to describe various technical design aspects of the project including code structure, dependencies, the build and configuration processes and the design patterns, especially from the perspective of developers and testers.

22.8.1 Code Organization

In this section we explore the code structure of SciPy. The code of SciPy is organized in folders in github. In the project, there are four main folders: benchmarks, doc, Scipy and tools. There a other files in the main project folder that are concerned with setup, gitignore, gitattributes, init files etc. We will explain the purpose of each main folder below:
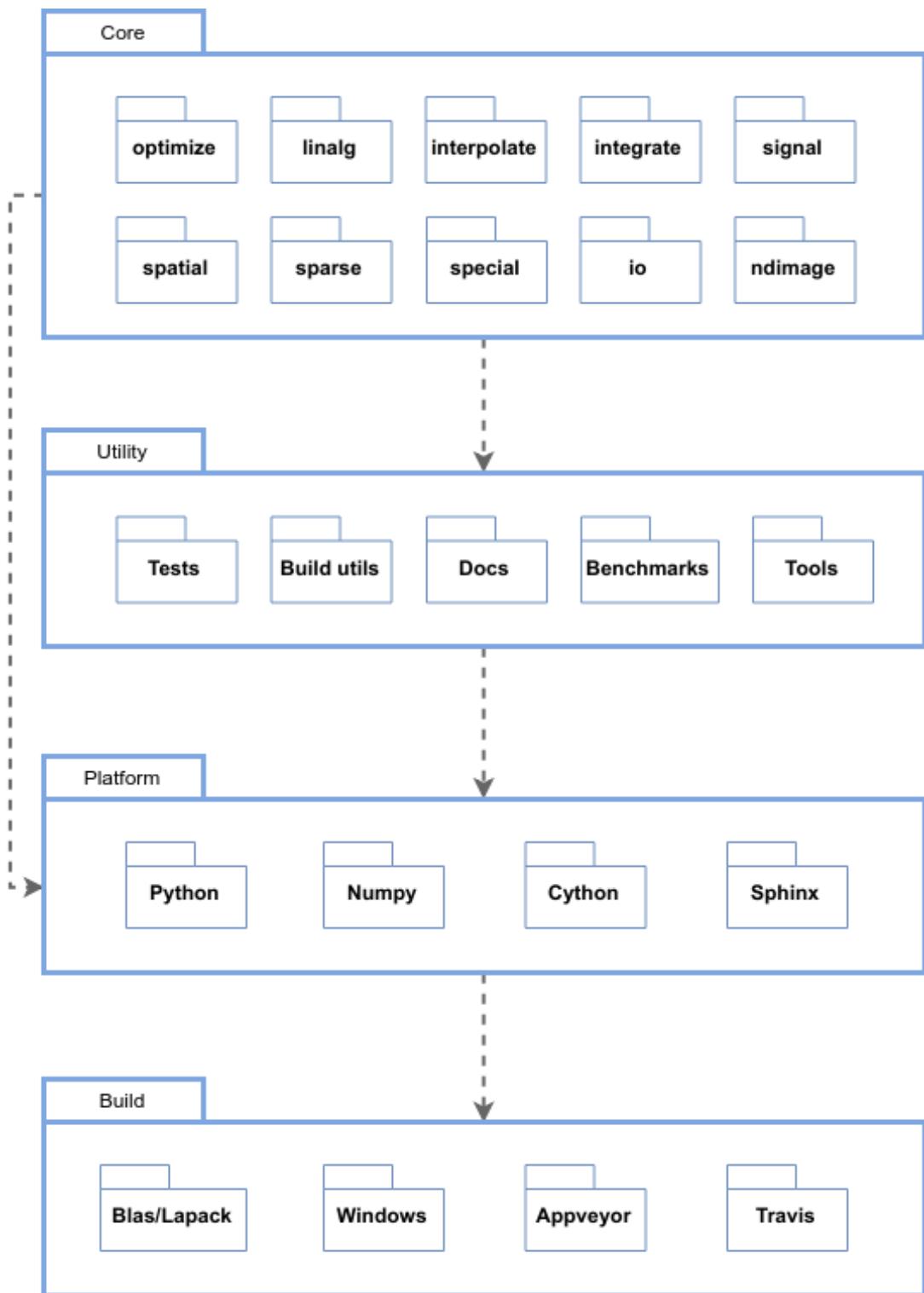
- **Benchmarks** This folder contains code for bench marking SciPy with Airspeed Velocity. This folder contains code for bench marking SciPy over its lifetime. Run time, memory consumption and value changes may be tracked over time. Benchmarks are written for interpolate, optimize, filtering and linear algebra functions.

- **Doc** This section contains the code used to generate the documentation. It contains documentation of each release. It contains a release folder that contains release notes of each SciPy release and explains the new features and changes, the deprecated features and backward incompatible changes. The source folder contains the code to generate the documentation. It also contains a tutorial folder that consists of example uses of SciPy. SciPy releases new versions as version number x.x.x and the release notes accompany each release and summarize all the files changed and functionality added or removed.

- **Scipy** This folder contains code for all functionality of SciPy. It contains a library folder that handles all dependencies and version checks. It contains each module of SciPy in a separate folder: integrate, optimize, linalg, signal etc. Each module contains its own test folder and its source code.

- **Tools** This folder contains code for CI (continuous integration), build scripts for Windows, installer file for MacOS, code to generate C files of python files and other helper files that are used in running tests and installation. It also includes scripts for GitHub like getting author names who contributed to the release.

Figure: Code structure of SciPy. The code is organized on github in folders. The sub directories are shown for only one level.

22.8.2   Module Structure Models

In this section we focus on the organization of the Scipy source code and group the code modules into layers of abstraction. We noticed that the source code can be organized into four layers based on its usage and functionality. These layers are: Core, Utility, Platform, Build.

Core

optimize	linalg	interpolate	integrate	signal
spatial	sparse	special	io	ndimage

Utility

Tests	Build utils	Docs	Benchmarks	Tools

Platform

Python	Numpy	Cython	Sphinx

Build

Blas/Lapack	Windows	Appveyor	Travis
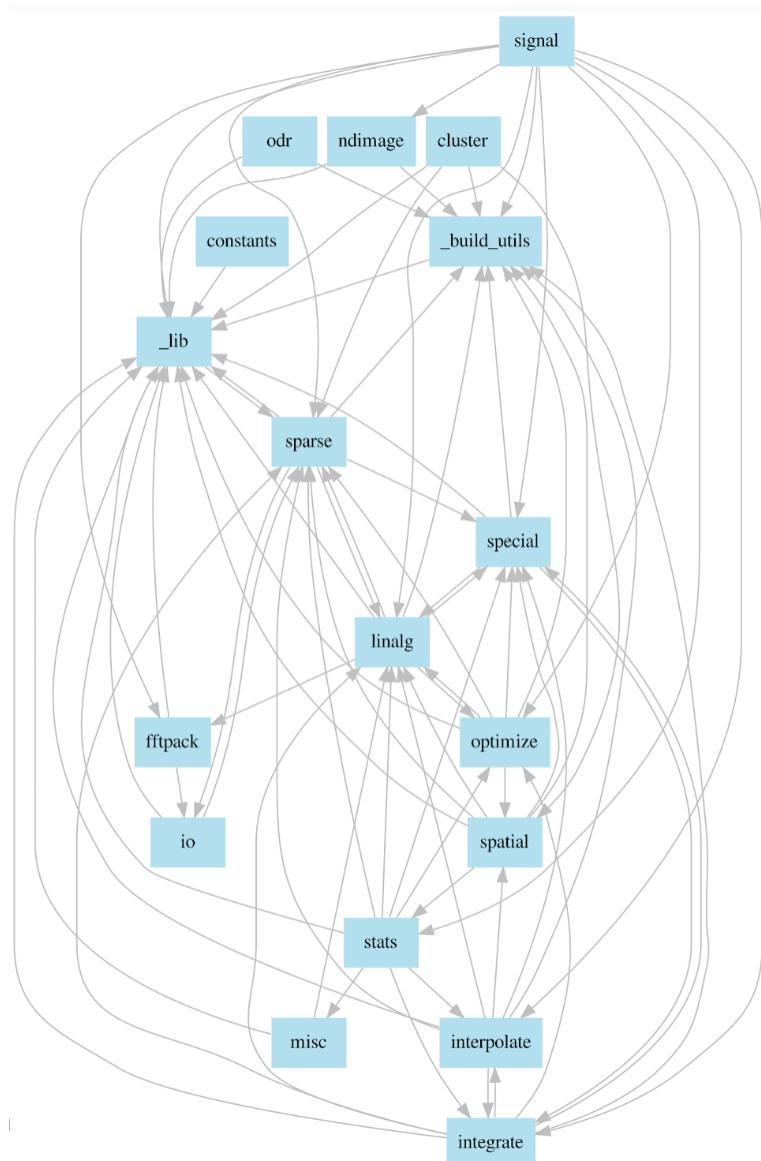
Figure: Diagram of module structure of SciPy. It can be described by a 4 layer structure based on the functionality of modules

- **Core Layer:** This layer consists of the main functionality of SciPy. It consists of modules that were present in SciPy folder of the repository. They contain implementation of all methods. It consists of optimize, linalg, interpolate, integrate, signal, spatial, special, io, sparse and ndimage. Here we have only shown the main modules of SciPy core. It consists of other modules as well like fftpack, constants, cluster and many more.

- **Utility layer:** This layer consists of modules that are utilized by the core layer and helper files. It consists of code for generating documentation, examples and tutorials, code for testing and bench marking against previous releases. It depends on the Core module and uses the methods from core module for testing and bench marking.

- **Platform layer** This layer consists of packages that are required by SciPy to run. SciPy needs Python, numpy, cython and sphinx (for documentation, optional). This module consists of the external libraries and dependencies. Dependencies are the programs or libraries that a user has to install in order to build/test a package. SciPy has minimum dependencies so that the user can easily perform the required tasks without any hassle.

- **Build layer** This layer consists of build modules. It consists of code to check for libraries and dependencies, get the correct versions according to system. It also contains installers and build scripts for Windows, Travis.yml file and Appveyor to build and test projects with continuous integration support. It also contains build and install instructions for BLAS/LAPACK which are recommended for SciPy. For CI support, SciPy relies on Travis and Appveyor.
  The support for multiple platforms is a primary reason for using two different CI services. It uses Travis for its support on OSX and Linux and Appveyor for windows.

22.8.3   Internal dependencies in core layer:

The modules in core folder are dependent on each other and every modules uses functionality implemented as part of other module(s) . The module level dependency has been analyzed and shown as a graph. Modules lib and build_tools contain utility and helper functions and are therefore imported by almost all other modules.

Figure: The module level dependencies in SciPy. It can be observed that the modules are heavily dependent on each other. We have only shown the internal dependencies of modules here and ignored the standard python libraries that are imported within the files.

22.8.4 Common Design Models

In this section, we describe the common designs used in the development of SciPy.

- **Design standardization** SciPy uses uniformity of style in its coding to make sure that the code is understandable. It follows the standard python guidelines for code style PEP8. It encourages

contributors to make sure that their code conforms to PEP8 standards and to use PEP8 package style checker. For new functionalities, SciPy follows a two level structure which means new function should appear as *scipy.submodule.my_new_func**.* Each module has a **init**.py file where public functions are imported and private functions and classes have a leading underscore (_) in their name.

- **Testing standardization** Every module in SciPy core is accompanied by its test folder that consists of unit tests that cover all functionality. For new code, testing guidelines are provided in Numpy testing guidelines which SciPy follows. For testing, SciPy uses the testing framework from numpy which uses Pytest framework and nose test suite. SciPy provides contributor guidelines to write their tests and encourages writing unit tests to exercise all the code they are adding. SciPy uses Travis and Appveyor for continuous integration and integrated with github to run tests on every pull request. This makes sure that the code does not run into any exceptions or errors.

- **Third Party libraries** SciPy relies on third party libraries for some of its functionalities. It uses python package Numpy for calculations. It also uses BLAS functions in its linalg module.

22.8.5   Programming Languages

In SciPy there are in total 2229 files. The code is written in multiple programming languages and the distribution is shown below (analysis performed using CLOC). Most code is written in C and Python.

Language	Files	Blank lines	Comment	Code
C	322	38125	162361	296772
Python	638	59108	103970	158677
Fortran 77	459	4238	74285	81899
Cython	41	5954	7998	32748
C++	19	2007	7493	22375
C/C++ header	117	3012	6240	14011
JSON	2	14	0	1455
TeX	2	131	161	1168
YAML	5	43	84	564
Others (css, markdown, html, TOML, make, matlab)	16	159	109	555

22.9   Deployment View

The development view of SciPy shows how the program is expected to perform at run-time.The SciPy is not a stand-alone of software. The view shows the dependencies and hardware and software requirements of SciPy.

22.9.1   Third-party Software Requirements

Dependencies are the programs or libraries that a user has to install in order to build/test a package. SciPy has minimum dependencies so that the user can easily preform the required tasks without any hassle, such as

run-time dependencies:	build-time dependencies:
Unconditional Numpy	Numpy; Cython (for development versions)
Conditional (test suite); asv (benchmarks); matplotlib (functions that can produce plots); Pillow (image loading/saving); scikits.umfpack (optionally used in `sparse.linalg`); mpmath (used for extended tests in `special`)	- setuptools; wheel (`python setup.py bdist_wheel`); Sphinx (docs); matplotlib (docs); LaTeX (pdf docs); Pillow (docs)

22.9.2 Working environments

As SciPy runs entirely within a single system, and on this system it runs inside python. Most of its working environment is either highly simplified or defined by its hosting programs. The following list shows some of these environments.

Environments	Description
Python(x,y)	It is a python distribution which is based on **Qt and Spyder.** It is used by free scientits and engineers for develping software for numerical computations, data analysis and data visualization.
WinPython	provides compiler for scientific analysis (Mingw64) for Python 3.4 which is fully integrated with **Cython and Numba.**
IPython	**IPython** (Interactive Python) is build for Python which provides command shell for multiple programming languages, it has features for visualization,use of GUI toolkits,etc.
Anaconda	Anaconda a enterprise-ready, free open source Python distribution which is a cross-platform software. Whose goal is simplification packages for Mac OS X, Windows,and Linux users.
Enthought Canopy	The free and commercial versions include the core scientific packages. Supports Linux, Windows and Mac.
Spyder IDE	Provides working environments on Windows and Ubuntu; Py2 only.
Microsoft Visual Studio	A free, rich IDE that supports Anaconda and Python. It also supports CPython, IronPython, the IPython REPL, debugging, profiling, Git and GitHub. Built-in languages include C,[6] C++, C++/CLI, Visual Basic .NET, C#, F#,[7] JavaScript, TypeScript, XML, XSLT, HTML, and CSS. Visual Studio Code is also available, which is a code editor with debugger. It is supported on Linux, Windows and MacOS.
Enthought Canopy	it is an analysis environment that includes Enthought's Python distribution which also has a analysis desktop with text-editor, code-checker and an IPython console. Enthought Canopy also includes TraitsUI, Pyface,Enable,SciMath etc on all platforms.
IEP	IEP is used for interactivity and introspection, which makes it very suitable for scientific computing. It is very simple to use a cross-platform Python IDE. The editor and the shell,are the two main components which also includes plug-in tools for the programmer use. Few examples are structure, project manager, interactive help, workspace,etc.
Pymacs	A Emacs tool allows users to communicate between Emacs Lisp and Python.

Environments	Description
Plotly	It is a online tool on Python environment for which is used for data exploration and graphing. Plotly contains a command line which allows users to store and share Python scripts.
Pyzo	Open and free distribution on Anaconda and the IEP interactive development environment. It is supported for Linux, Windows and Mac.
Wakari	A browser based Scientific and Technical Computing tool. users can create and share workflows, IPython notebooks, plots, and applications on cloud which also features Anaconda- Big-Data Python distribution.

22.9.3   Basic Versions required

During the initial installation the following distributions of packages should be included. The (x.x.x) indicate the required versions for the dependencies to run on SciPy.

Dependencies	Version required
Python	(2.x >= 2.6 or 3.x >= 3.2)
NumPy	(>= 1.6)
SciPy library	(>= 0.10)
Matplotlib: dateutil; pytz; Support for at least one backend	(>= 1.1)
IPython (>= 0.13): pyzmq; tornado	(>= 0.13)
pandas	(>= 0.8)
nose	(>= 1.1)
Sympy (>= 0.7)	(>= 0.7)

22.9.4   Running Code Written In Other Languages

Below is a list of software platforms which helps wrapping code of different languages.

- SWIG: SWIG is a software that primarily connects programs written in C and C++ with a wide range of high-level programming languages. It also supports scripting languages such as Javascript, Perl, PHP, Python, Tcl and Ruby.
- Boost.Python: a C++ library which enables seamless ability to transform between C++ and Python.
- F2PY: Mainly used to warp Python and Fortran languages. F2PY is a Python extension tool for creating Python C/API modules from signature files (or directly from Fortran sources).
- matlab: Has interfaces for Python by treating it as a computational tool. For information about how to interface with Python from MATLAB, visit this link here.
- pythoncall:it connects MATLAB-to-Python which runs a Python interpreter inside Matlab and allows transferring data between the Python and Matlab workspaces.
- ctypes: a package which allows users to create and manipulate C data types in Python, and to call functions in dynamic link libraries/shared dlls. It wraps these libraries in pure Python programming language.
- railgun: ctypes utilities for faster and easier simulation programming in C and Python.

- rpy2: a very simple, yet robust, Python interface to the R Programming Language. It can manage all kinds of R objects and can execute arbitrary R functions (including the graphic functions). All errors from the R language are converted to Python exceptions. Any module installed for the R system can be used from within Python.
- mirpyidl: it provides a library to call IDL (Interactive Data Language) from python. Allows transparent wrapping of IDL routines and objects as well as arbitrary execution of IDL code. It utilizes connections to a separately running idlrpc server.

22.9.5 Minimum Hardware Requirements

Below listed is the basic hardware requirements for running SciPy successfully.

- Disk space: 1 GB
- Operating systems: Windows* 7 or later, MacOS, and UNIX-based systems
- Processors: Intel Atom® processor/Intel® Core™ i3 processor or higher

22.9.6 Making a SciPy release

Manger is the main actor in release of a new SciPy release, below are the steps followed:

1. Schedule is proposed and released on the Scipy-dev mailing list.
2. maintenance branch is created for the release.
3. Tagging
4. Building is released for artifacts such as sources, installers, docs.
5. Upload the release artifacts.
6. Announcement is released.
7. Port release notes and scripts are built onto master.

22.10 Technical Debt Analysis

In this section we analyse the technical debt of SciPy. We make use of an automatic tool for code quality analysis. The subsections respectively describe the debt identified in the system with regard to defects, code-style, testing and documentation. The following metrics were analysed:
(Note: In this section, we discuss the relevance of the metrics with SciPy and values noted from our code analysis. Intricate details have been explained in the Appendix C)
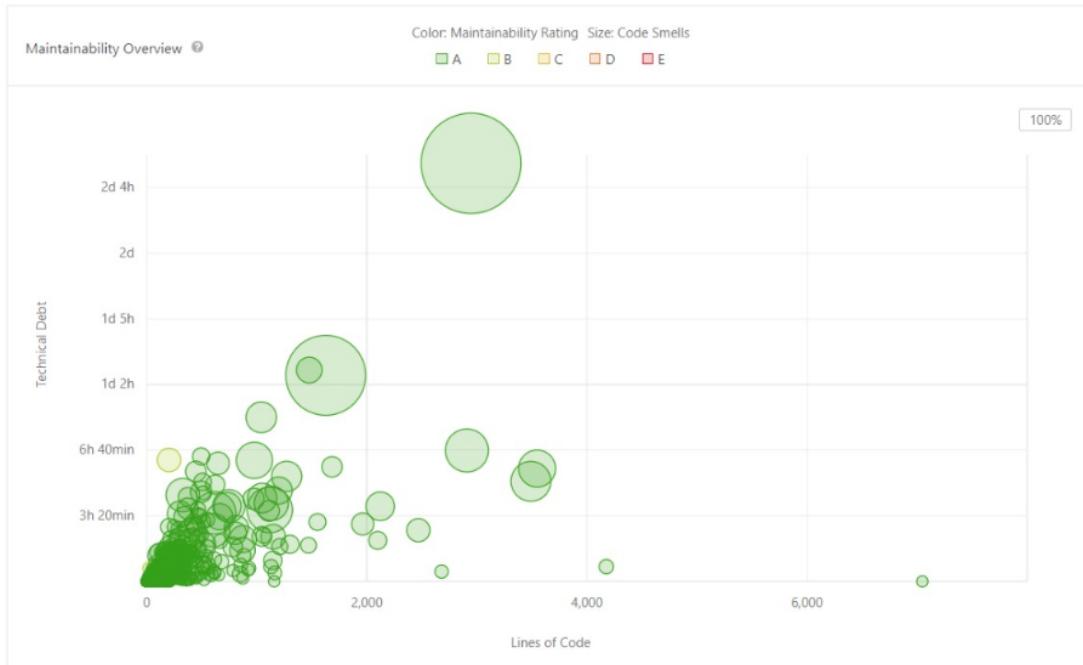
22.10.1 1. Maintainability

Maintainability is focused on code smells, which is a maintainability-related issue found in the code analysis [1].

1. **Code Smells** Code Smells refers to any symptom in the source code of a program that possibly indicates a deeper problem. [2]

For Scipy, we noted **4,373** Code Smells from the results of static analysis. We used Quality Profile evaluation to be the default Sonar Way evaluator. Additionally, the benchmarks we reported against default Sonar Way benchmarks. The debt that was reported for the code smells were estimated 56 days. The maintainability ratio was grade with **A**. The reported technical debt ratio for Scipy was 0.5%

The figure below depicts the results obtained from the test using SonarQube.

Figure: Maintainability Overview

22.10.2   2. Code Complexity

Code Complexity or Cyclomatic complexity is a quantitative measure of the number of linear independent paths through a program code. For SciPy, the code complexity was performed using **Pylint.** Pylint analysis the code at a file level and also performs an integrated complexity check with adherence to the standard developed by T.J McCabe in "A Complexity Measure." Additionally, Pylint provides a rating based on the analysis performed on a scale of 10. For the SciPy project, we received a rating of 2.67/10. As a remediation measure, we have listed possible ways to obtain better scoring of Pylint's code complexity analysis [3].

22.10.3   3. Code Duplication

This section describes the amount of code blocks that were found to be duplicated during the static code analysis. [4] In Scipy, we observed 280 code blocks that were copied, which amounted to roughly 1.3 % of the total code base. The Code Duplication was performed using SonarQube. The results of code duplication is represented in the figure below where the size of the bubbles represent the duplication blocks of code.
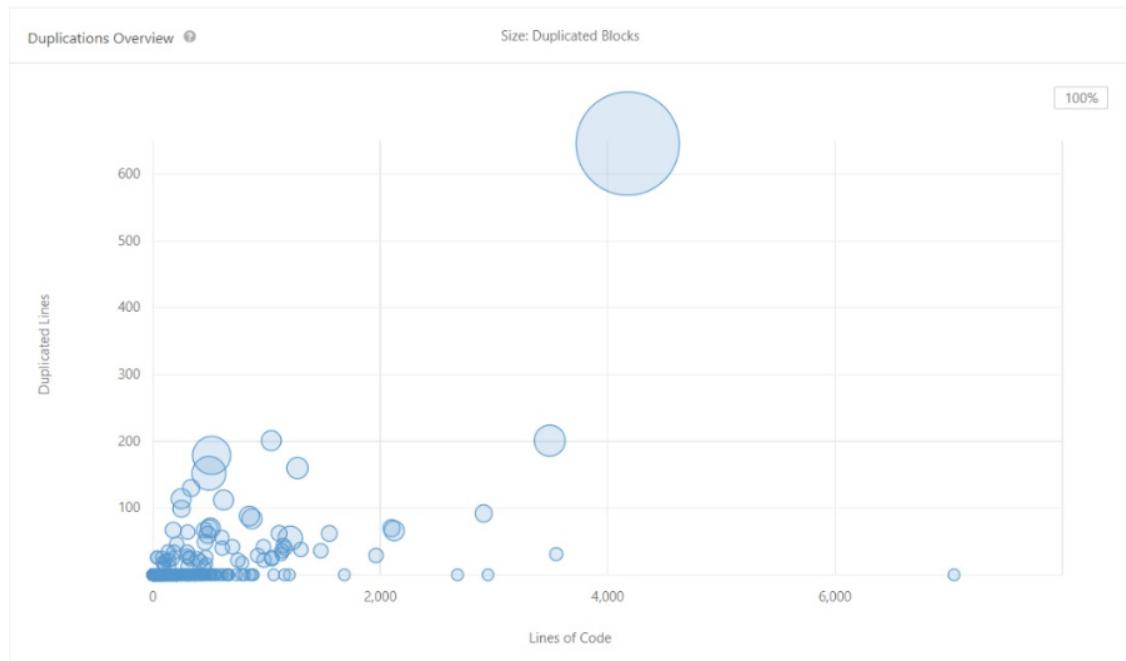
Figure: Visual representation of Duplication

22.10.4   4. Reliability & Security

1. **Reliability** The source of reliability in code is the ability to produce a predictable outcome given wildly varying inputs. [5] For Scipy, the Reliability Ratings graph is represented with a bubble graph. Additionally, the graph showcase the number of bugs reported while performing the code analysis. We have used SonarQube for reliability reporting
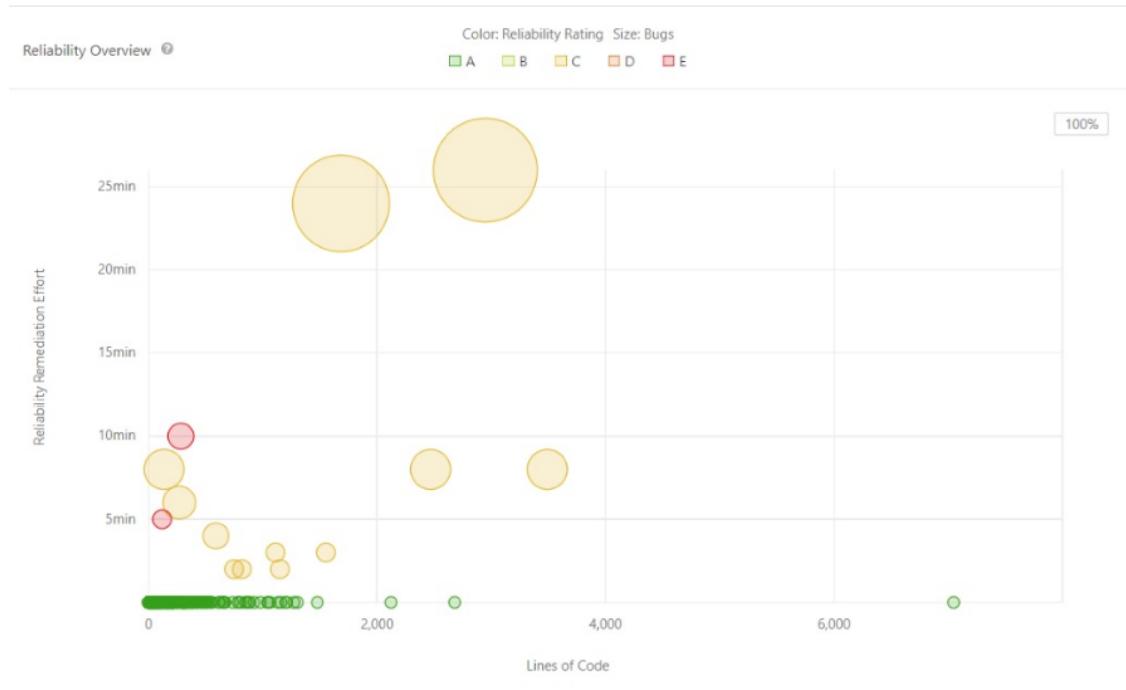
Figure: Visual Representation of Code Reliability

b. **Security** Static analysis looks for security vulnerabilities in source code by referencing the most common vulnerabilities as defined by OWASP. [6] The Figure below depicts the Security related vulnerabilities obtained as part of our analysis. We note that there are no security vulnerabilities for SciPy. We have used SonarQube for security vulnerability reporting
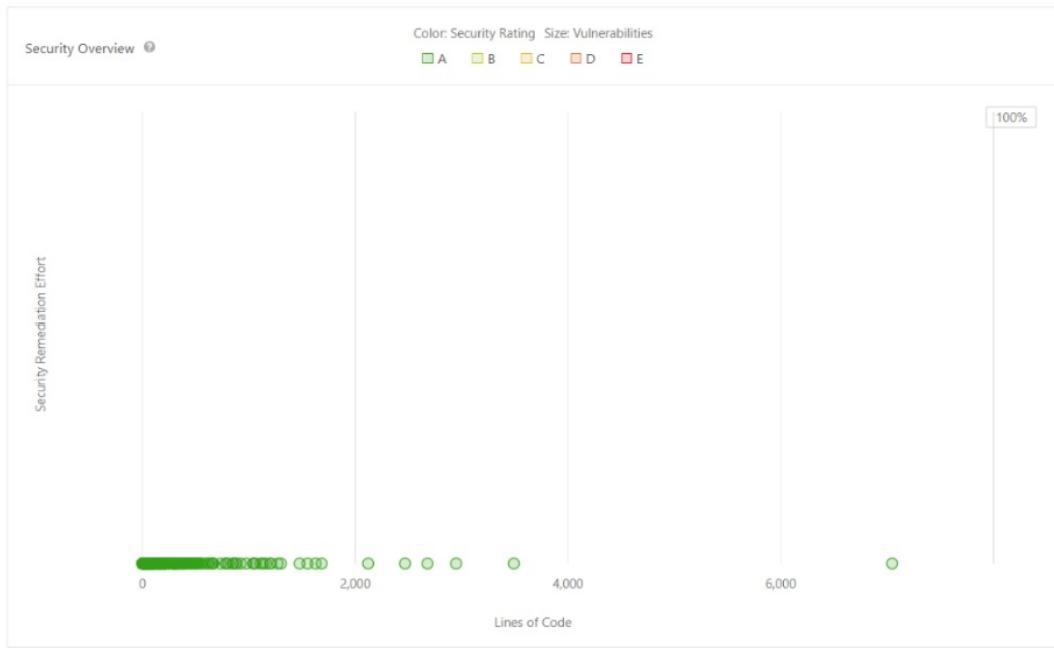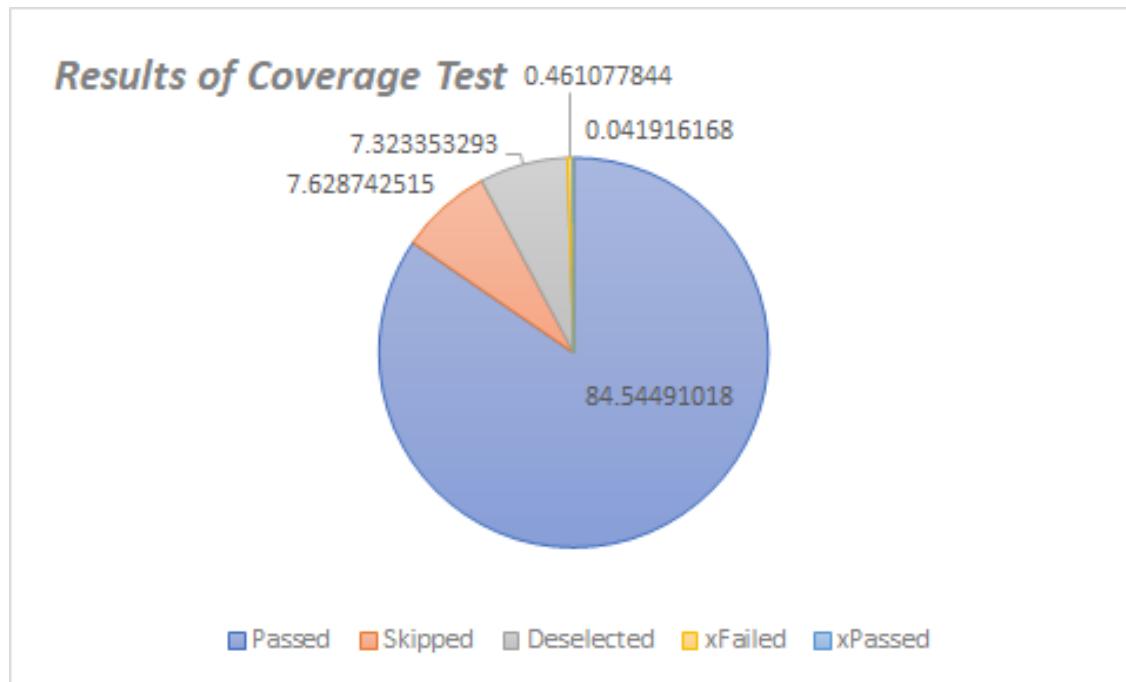
Figure: Visual Representation of Security Analysis

22.10.5   5. Code Coverage Tests

Code coverage describes the degree to which the source code of a program is executed by a particular test suite [7]. For Scipy, as part of the code coverage task, missing test coverage's long-term risks are analysed. We have used native SciPy testing suite for our code coverage tests.

A visualization has been made from our results below.
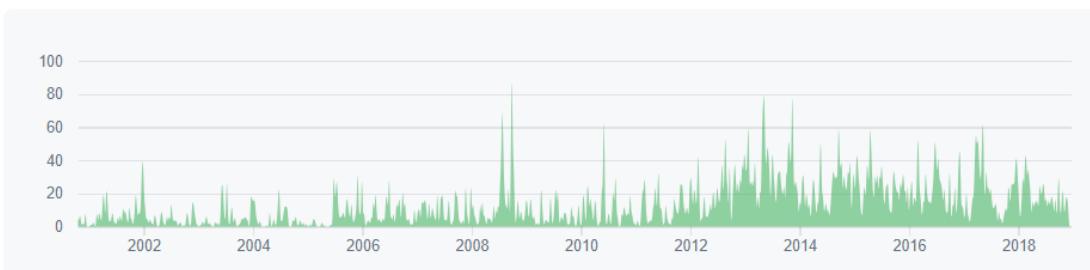
Figure: Results of Coverage Tests

The run-time of the test was 543.9 seconds. The important metrics to note here would be the passed legend which represents code pass as part of our testing and xFailed which represents code failure which is reported at approximately 0.5% of our code base.

22.10.6   6. Dependency of Individuals to SciPy
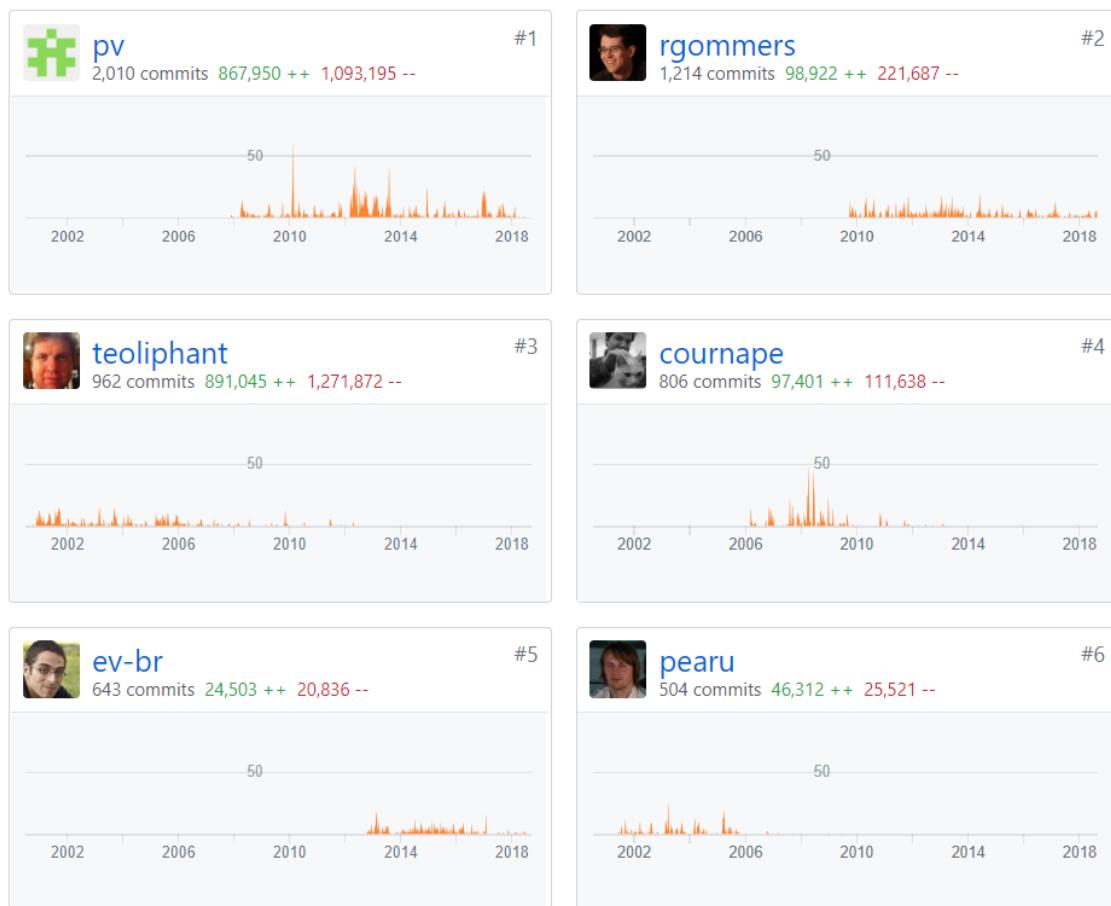
In this section, we analyse the dependencies of individual contributors of Scipy. If there are more sole responsibilities assigned to a select people of the Scipy project, it is implicitly considered to be high risk as the absence of these individuals may lead to failure in the progress of Scipy.

The figure below reports the overall activities of the contributors from the start of SciPy.

Dependency graph of all the activities of contributors from inception of Scipy

The figure below shows the top 10/100 contributors of Scipy. The contributions to master, excluding merge commits are depicted here. With the contributors being:
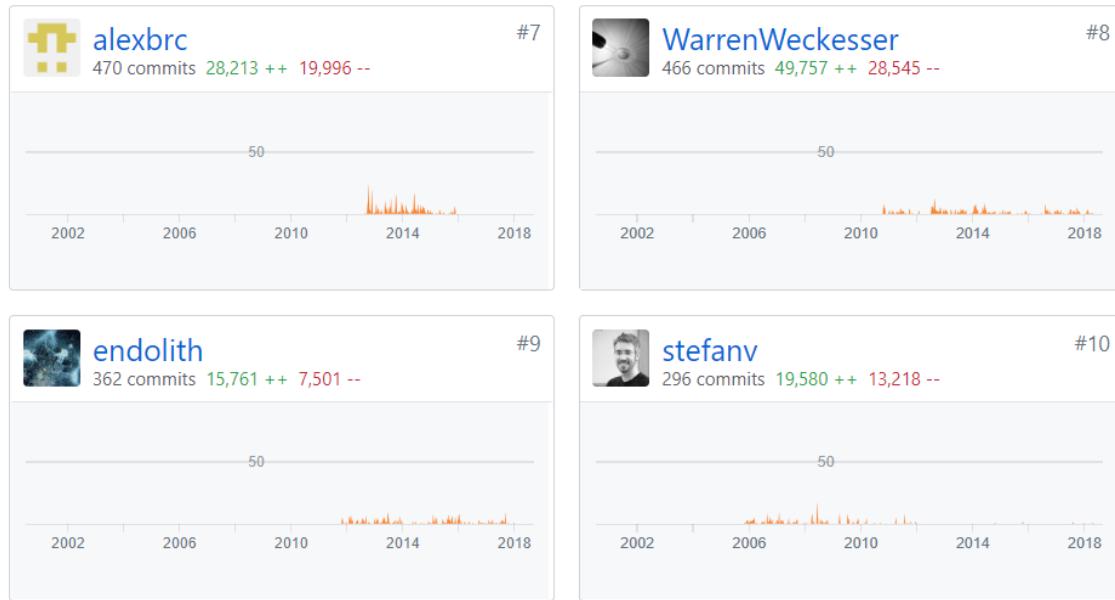
Figure: Top 10 contributors to SciPy

For SciPy, there has been fluctuating yet distributed dependency graph of Contributors. Hence, this does not affect the progress in lack of any significant-individual dependency for SciPy.

22.10.7   7. Evolutionary Perspective

In this section we discuss, the evolutionary perspective of SciPy, its past and present transitions and the present version usage. Additionally, we address documentation debt.

a. Documentation Debt Scipy is a massive project with over 160,000 lines of code. While the code itself had appropriate comments in most of the sections, we were unable to find any documentation that detailed the architectural details and design principles followed in the project. Also, given that Scipy has an extremely active contributor base, the need for strong documentation was very evident in our study.

b. Evolutionary Diagram - Code Frequency As part of reporting the Historical Analysis of Technical Debt we have used the Evolution diagram which showcases the code additions and deletions of Scipy over a time stamp as per 1 year. The reporting is done every year in the graph below since the inception of SciPy in 2001.
The below figure is the code frequency of Scipy activity.

Figure: Code Frequency

The peaks in green legend refer to the additions that were reported and made to commit while the red refers to the deletions that were performed from a code removal standpoint. It was noted that on 2007-2008 saw the highest addition and deletion of code implementations for Scipy. However, asides this, there has been a healthy code addition/deletion report graph for all years.

c. Versions SciPy has 103 releases in total and the present version release is v1.2.1 (on February 8, 2019).

d. Testing Debt In Summary, we used the following tools for Code Analysis for reporting the metrics:

1. SonarQube with Travis CI integration - Maintainability, Code Duplication, Reliability & Security
2. Pylint - Code Complexity
3. SciPy Native test suite - Code Coverage
4. SciPy Insights - Evolutionary Perspective, Dependency of Individuals to SciPy.

As part of contributing to reduce the testing debt, we identified critical issues from testing of technical debt and raised PR requests with SciPy team for approval. The possible methods to reduce testing debt and our contributions to SciPy have been discussed in Appendix D.

22.11   Conclusion

SciPy is currently the most popular and widely used tool for students and data scientists for scientific computing, mathematics and engineering. The course provided an exciting opportunity for us to perform an in depth analysis of one of the most famous python libraries in its domain. Listed below are some key activities we performed as a part of our curriculum.

- Identified key stakeholders and drafted a contextual view.
- Performed technical analysis on the SciPy library via deployment and development perspectives.
- Performed Static Code Analysis on SciPy's code repository using automated tools
- Performed Pull Request (PR) analysis and defined an appropriate Codification Standard for merged and unsuccessful PRs.
- Made a contribution to the project in form of describing one of the fundamental classes based on an open issue. Based on technical debt analysed, some contribution is still in process and we hope to make more contributions to the library we love using.

Based on these activities, we were able to draw the following conclusions.

- SciPy has a high degree of reliance on the open community of contributors. The participants own significant responsibilities on the library.
- It caters to wide set of user-group both in scientific as well as commercial communities.
- It builds on other libraries in the scientific computation suite and offers abstraction of a wide range of scientific functionalities.
- While we did notice significant technical debt and possible improvements, SciPy is under active development support by its strong community.

22.12   References

1. Nick Rozanski and Eoin Woods. Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives. Addison-Wesley, 2012, 2nd edition.
2. SciPy Issues(https://github.com/scipy/scipy/issues)
3. SciPy documentation (https://docs.scipy.org/doc/scipy/reference/roadmap.html)
4. Arie van Deursen and Maurício Aniche and Joop Aué (editors). *Delft Students on Software Architecture (DESOSA)*. https://delftswa.gitbooks.io/desosa2016/content/, TU Delft, 2016.
5. SciPy Donations webpage: https://www.scipy.org/scipylib/donations.html ## Appendix A: Pull Request Analysis

The table below summarizes the context of the pull requests we analyzed.

22.12.1   Accepted requests

ID	Proposed by	Nr. part.	Active discussions by	Started on	Merged on	Tags
4374	@Sturlamolden	8	@ewmoore, @jakevdp @rgommers, @ogrisel	Jan 7, 2015	May 10, 2015	Enhancement, scipy.spatial

ID	Proposed by	Nr. part.	Active discussions by	Started on	Merged on	Tags
4890	@Sturlamolden	8	@ewmoore, @jakevdp @rgommers, @pv, @rainwoodman	19 May, 2015	22 Nov, 2015	Enhancement, scipy.spatial
8357	@mikofski	16	@tkelman, @person142 , pvanmulbregt , @rgommers , scopatz, @andyfaff, @jaimefrio	Feb 2, 2018	Jun 25, 2018	maintenance, scipy.optimize
335	@js850	6	@josef-pkt, @diax, @rgommers	Oct 9, 2012	Feb 3, 2013	not defined
3174	@ev-pr	9	@rgommers @pv @mathew-brett	Dec 27, 2013	Oct 21, 2016	enhancement, scipy.interpolate
7616	@ghost	8	@ larsoner, @ matthew-brett @pv, @ rgommers,	Jul 17, 2017	Aug 12, 2017	Build issues, enhancement
6326	@nmayorov	15	@ nmayorov, @ aarchiba, @pv, @ bmcage, @ev-br	Jun 28, 2016	Apr 21, 2017	enhancement, scipy.integrate
3717	@larsoner	8	@larsoner, @coveralls, @argriffing, @endolith, @rgommers, @pv	Jun 06, 2014	Jan 21, 2015	enhancement, scipy.signal
4021	@insertinteresting8		@larsoner, @coveralls, @argriffing, @endolith, @rgommers, @pv	Sep 24, 2014	Mar 27, 2015	enhancement, scipy.linalg
5608	@stsievert	13	@larsoner, @ stsievert, @ jaimefrio, @ tacaswell, @rgommers, @pv, @ WarrenWeckesser, @endolith	Dec 15, 2015	Jul 20, 2016	enhancement, scipy.signal

22.12.2 Rejected requests

ID	Proposed by	Nr. part.	Active discussions by	Started on	Merged on	Tags
6725	@alvasorg	7	@nmayorov, @josef-pkt, @ev-br	Oct 25, 2016	Feb 9, 2017	needs-decision, wontfix
6895	@nils-werner	6	@pv, @larsoner	Dec 28, 2016		scipy.signal,wontfix
5301	@giorgiop	6	@jaimefrio, @argrigging, @ev-br, @josef-pkt	Oct 1, 2015	Jab 18, 2016	defect,scipy.spatial,wontfix
3044	@bbudescu	3	@pv, @rgommers	Nov 7, 2013	Sep 17, 2017	Build issues,needs-work

ID	Proposed by	Nr. part.	Active discussions by	Started on	Merged on	Tags
8454	@samyak0210	7	@peterjc, @ilayn, @pv, @ev-br, @rgommers	Feb 21, 2018	Jun 11, 2018	maintenance, scipy.integrate
7033	@ashwinpathak20	4	@ashwinpathak20, @person142, @ev-br, @rgommers,	Feb 12, 2017	Feb 16, 2017	Documentation, scipy.special
3893	@WarrenWeckesser	5	@ coveralls, josef-pkt, @pv, @ WarrenWeckesser, @rgommers,	Aug 22, 2014	Dec 02, 2014	Documentation, maintenance
5226	@WarrenWeckesser	3	@ argriffing,, @pv, @WarrenWeckesser.	Sep 04, 2015		defect,needs-work,scipy.special
4379	@juliantaylor	10	@larsmans, @ jaimefrio, @ev-br, @WarrenWeckesser	Jan 08, 2015	Jan 21, 2015	enhancement,scipy.spatial

The analysis and our theories about each pull request are explained below.

22.12.3   Accepted requests

ID	Contributions	Decision
4374	This PR was mostly related to code enhancement in the spatial package. The main additions in the initial PR were: Make the kd-tree data structure inspectable and walkable from Python; Add support for pickle; Removed memory leaks in cKDTree.query; Use PyMem_Malloc instead of libc malloc; Add an option for creating balanced trees; Add an option for compacting/shrinking tree nodes; Clean op the coding style; Change language to C++; Reimplement cKDTree.query in C++; Release GIL in cKDTree.query. It started out as work in progress and the tests were added later. One of the most discussed topic in comments was to use the scikit-learn implementation of kd-tree but the author pointed out the differences between both and this implementation was comparatively faster. The added functionality was tested on Linux, windows and OSX.	This PR was merged. According to our analysis, the performance of the kd-tree method implemented was significantly better than the previous ones available. This was one of the main reasons for merge. The enhancement also passed all of the tests. It only gave an error on MinGW-w64 past gcc 4.x but that was due to another dependency that needed to be fixed separately so the decision to merge was taken.
4890	This PR was created after debasing #4860 which was related to porting the remaining cKDtree query methods to C++. Moving to C++ allows for significant speed improvement. Moving to C++ resulted in 6x faster performance as mentioned in PR #4860. Some initial work is also found in #4850. All the query and build methods and many conditional branches are converted to C++. There was a significant improvement in two query methods. Pickling was added in python 2.	This PR was a follow up to the cKDtree implementation work and work was related to porting the methods to C++ to improve performance. The tasks still pending were testing on Windows, code review, bug fixing and using a different method in one of the functions. Updates for these tasks were given in the comments and discussion. An important point of discussion was to benchmark all releases of cKDtree and this was done in a separate repository. There was some performance regression for few methods compared to an earlier release but those were resolved in later commits. After passing all the builds and tests successfully, it was merged. According to us, the decision to merge was most likely made because of the 6x faster performance that makes this a significant improvement.

ID	Contributions	Decision
8357	This PR was created to address the issues in #8354. In #8354 a custom vectorized newton method was proposed. This PR adds tests for newton method with arrays and checks newton step deltas and checks for zero derivatives. This PR also addressed #7242.	Most of the discussion in this PR was related to coding practices, suggestions on how to improve the code and explanation of the programming decision by @mikofski. The vectorized implementation was bench-marked against other alternatives. The bugs identified through the discussion were debugged and fixed. Many code fixes were implemented for example, fixing the exceptions, wrong assumptions about function arguments etc. The performance degradation cases were successfully addressed and test cases were added. This PR improves the performance for those cases where a vectorized approach is needed and supports the scalar version too. In our opinion it was merged because it is a useful change that provides a vectorized implementation of newton's method which didn't exist before.
335	This PR introduces the implementation of basinhopping global optimization algorithm. It used the minimizers already implemented in scipy.optimize. Basin hopping is a random algorithm which attempts to find the global minimum of a smooth scalar function of one or more variables. It is a useful approach for non linear optimization.	The initial comments to the PR were about coding style and suggestions to fix it, for example: more descriptive variable naming, spacing etc. Other comments were about understanding the algorithm implementation. It was suggested to add more test cases. This algorithm was also merged. We think it was merged because it was a new feature that was a useful addition to scipy.optimize.

ID	Contributions	Decision
3174	This PR implements a base class for evaluation of b-splines without fitpack. It implemented a base class which could be used through API directly and in the interpolation/fitting algorithms. It also contained tests to verify the implementation.	In the discussion, updates regarding implementation were given. The behavior of class was updated to be consistent with the definition in literature. The other comments were by reviewers asking about the purpose of some lines in the code and the reasoning behind adding them, suggesting alternatives or suggesting changes to improve the code structure, improve the implementation and make code more readable and conform to the standards used in rest of the code base. There were also considerations about how to replace the existing spline implementations with this implementation. The implementation was initially slower than fitpack so it was investigated how to improve that and updated in later commits. The implementation was given a needs-work label by @ev-br because the algorithm was for creating periodic splines was naive and needed a fix with linear algebra. And the periodic spline part was removed to be able to merge this. This PR also deprecated splmake/spleval. The final performance was comparable with fitpack. The PR was closed and reopened for testing. The issues of revierwers were addressed in new commits and changes made were approved by @pv and merged into master. It was merged as it was an enhancement in scipy.interpolate and offered an implementation without one dependency (fitpack) with comparable performance as the implementation with fitpack.
7616	This PR is about enabling builds on to Appveyor for effective test integration. Issue was encountered when the fotran code had C code calling usage. As solution, all the DLLs (with OpenBLAS and mingw dlls) were separately stored. Was stored under tool/ directory. Additionally, due to verbosity, numpy.distutils was taken up for cleanup.	This PR was merged. According to our analysis, the performance of Appveyor on 32 bit implementation gave lot of errors and they worked on that as the major PR upgrade. Although the initial issue was stderr malfunction which was eventually solved, the PR thread was continued with 32-bit implementations.

ID	Contributions	Decision
6326	In this pull request, @nmayorov, has initiated a request and suggestive implementation to the ODE class belonging to "scipy.integrate.ode". A monolithic implementation with solutions to address the issues of no continuous solution output, (i.e. which can be evaluated at any point with the same accuracy as the values computed at discrete points) and absence of event detection capabilities are proposed eventually.	At the start of the PR discussions, there are conversations from @pr, @aaricha and the author of the PR on the scientific aspects of the ode class and its implications. There are clarifications provided and example cases are presented such as the RK45 method, DOPRI5 etc. Upon discussions, it was found that the ODE solvers in Matlab were also of research prospect. @drhagen proposes an event detection mechanism to the ODE solvers in scipy. A code level change of the return parameter was changed to suite the functionality of the solver. As for the testing and compilation, the team encounters some issues and they are resolved in the same PR as comments. Upon testing, it is noted that there are issues in the test case failures and the troubleshooting steps are undertaken by doing code-level scans (manually) at each function library and class solvers. Finally, it is concluded that there is an option to include where the solvers switching between stiff and non-stiff solvers can be made. The code is created and tested, and the original design is retained. Additionally, the event callbacks are updated in accordance with the new code changes. A re-test is conducted. 1 out 2 tests only pass. However, they still go ahead with the merge and the contributor is complimented for the suggestion.

ID	Contributions	Decision
3717	In this pull request, @larsoner, has initiated a request from a previous PR - #2444 which discusses the addition of support for second order sections in scipy.signal. PR #2444 was re-opened along with PR and closed after successful merge in Jan 21, 2015 of PR #3717. Here, the author proposes the following changes: sosfilt, which performs filtering using second-order sections. zpk2sos / tf2sos to convert to sos format. Wanted to remove tf2sos. users might have filters in tf form and think that going to sos will reduce numerical error. Forcing them to do zp2sos(tf2zpk(b, a)) might make them think twice about why there isn't a tf2sos function. *(seeking opinions)* cplxpair and cplxreal, which are helper functions for dealing with complex conjugate pairings.	@endolith, starts the conversation with the review of @larsoner's implementations and encounters minor travis errors which are eventually fixed at once. Additionally, as the review procedure continues, there are several errors that are reported such as functional issues of n_stages and n_sections, etc. Additionally, a critical error at review was reported: *Changes Unknown when pulling e4f4aa0 on Eric89GXL:sosic into* on scipy:master. This forms the basis of the discussion as it was attributed to continuous code failures. The author resolves the same. And the discussion on the test results are discussed. There are code changes being made in terms of logical attributions and not syntaxial or run-time relevance. The code is tried in many second order cases such as analog ordering cases, computational efficiency testing etc. Upon testing the code on possible mathematical test cases, a merge approval is provided by the contributors and the PR is merged. However, in May 24, 2016, it was reported that there was an issue recorded "scipy.signal needs a unified filter API" which is tagged to a PR #6137 which was created at a later stage and followed up to closure.

ID	Contributions	Decision
4021	In this pull request, the author has requested for support for the implementation of lpack and blas (which are mathematical/scientific libraries) for Cython via API referencing. In the base request comment the following have been identified, codified and implemented (the implementation was identified as work in progress by the author): *Add interface for lapack and blas routines.*; *Add working example in tests*; *Review signatures for correctness*; *Add function to get directory of scipy.linalg (similar to numpy.get_include)*; *Finish wrappers for 'gees' and 'gges' functions.*; *Add interfaces to the functions added in #3984 (wrapping blas routines)*; *Add this api to the documentation*	@pv identified an error "f2py._cpointer does not point to the wrapped fortran routines, but directly to the BLAS/LAPACK symbols" which is attributed to incorrect pointer referencing for ABI calls. Furthermore, a formulated list of cases of failure are identified and introduced in the forum. The changes are worked upon. The solutions are identified by changing the entry statements in entry wsdot(n,x,offx,incx,y,offy,incy) and altering call statement declarations. Upon rectification, a code review is performed by @ewmoore and has reported issue in calling at the interface of python scipy.linalg.lapack. The author worked on this assigning arithmetic float pointers in the calling interface thereby making them safe on all platform executions. Minor code beautification procedures are discussed and brought up as issues which are concurrently resolved. Another PR #4105 BUG: Workaround for SGEMV segfault in Accelerate is referenced to this PR. The relevance is explained. A cythonize script is written for adding a submodule named cython to preclude the use of these wrappers in scipy.linalg itself. This is being reviewed by Scipy contributors and the resultant warnings, errors are rectified in successive conversations. A major build failure with Cython 0.20.2 was reported by @rgommers. Additionally segmentation faults (segfaults) were also reported alongside the build failure. The root cause is performed and found that test_blas_pointers.test_wfunc_pointers that crashes, the other tests pass and also the segfaults belonged to different cynthon version. @rgommers performs a backtrace of the code and rectifies the errors with the help of fellow contributors. Asides, these major issues, there are minor clarifications seeked out with minor test failures eventually rectified in subsequent conversations. A complete rebase is done at the end and merged successfully!

ID	Contributions	Decision
5608	In this pull request, the author has extended the work of PR #2651 (this PR adds the keyword argument method to scipy.signal.convolve to choose the choose the convolution method. method can take vales 'auto', 'fft' or 'direct') and PR #1792 (faster convolution method, either the direct method or with fftconvolve.). In this PR, the author, has proposed a test failure and further details on impactful documentation for the above-mentioned parent PRs. A build failure was noted while using ffconvolve. The root cause of this issue was traced to passing on TravisCI without any adjustment. Later, the author conducted the test again with gcc and Homebrew and noted the ffconvolve to plot the points correctly.	@ jaimefrio begins the discussion with a review of the proposed fix. It was suggested to change the default aspects of ffconvolve to parameterized custom initializations which was agreed by several contributors to the PR. However, there was an observed error log that was reported by @rgommers while executing the same on a 32-bit system (Errors being test_convolve_method (test_signaltools.TestConvolve), test_rank3 (test_signaltools.TestCorrelateComplex192), test_rank3 (test_signaltools.TestCorrelateComplex192) and test_signaltools.test_choose_conv_method). However, the 64 bit systems did not report these error messages. Hence, the task of resolving these four error messages were taken up and discussed in the PR workflow. The author had proposed the re-worked code and was tested by the contributors and approved for appropriate functionalities. The PR was finally merged successfully.

22.12.4   Rejected requests

ID	Contributions	Decision
6725	This PR implements a method to calculate the numerical inverse of any invertible continuous function. The method used existing functions in scipy to solve the problem of finding the inverse of function under continuity and monotonicity conditions. There were certain conditions mentioned under which the algorithm failed to give accurate results. The functions should be continuous and strictly monotonic.	The discussions about this PR were about the motivation for adding such a function, explaining possible use cases. Suggestions about variable names and pointing out mistakes in the implementation (the intervals were not supported, global root finding was not well defined). The members of scipy were not in support of the idea of having a library function to numerically invert a given function. The approach was found to be flawed by the reviewers. Main reason was that the method failed to detect if a function wasn't monotonic and the choice of algorithm to find inverse is problem dependent and this algorithm isn't fit for that purpose.
6895	This PR aims to deprecate get_window and instead have people use the window functions directly. The current implementation of get_window() is very messy and consists of many conceptual problems. It adds functionality to pass a callable to get_window(), replaces all occurrences of get_window() and raises deprecationwarning when no callable is passed. It also removed string arguments.	The main reason for rejecting this PR was that the changes were not backward compatible. Most code is not actively developed and the users who obtained code from a source may not have the expertise to fix it. The new method does make things clearer but was ruled out due to backward incompatibility. There are also places where using string arguments is the clearest option so that made the string argument removal not useful. The benefits of the cost were not worth the cost and the PR was closed.
5301	This PR handles corner cases for correlation. Correlation is not defined for constant input vector but it was defined to complete the definition at the limits of vectors with zero variance. This implementation was also justified by the reasoning that if variance is zero, co variance is also zero and correlation distance is 1.	In the discussion, few coding changes were suggested (for example to change the iteration). There were also questions about whether defining corner cases made sense and that using NaN for corner cases made more sense. The reviewers were not convinced that returning 0 or any value instead of NaN for corner cases was correct. It was initially proposed to clarify the documentation by adding the limit behavior changes in doc strings in either this or new PR. The PR was closed because there was no work update and they made the decision to keep the current behavior.

ID	Contributions	Decision
3044	This PR provides a hack to build scipy on win64 with msvc9. It was proposed that this code could be used to make a true solution that works across platforms and compilers.	The functions were added manually in the build and that was considered a wrong solution. There were issues related to fortran that arise when using static libs from fortran code. The PR included match functions of C99 in msvc2013 but it was not clear why they were needed and suggested to look into npy_math to find the issue. During testing, python continued to crash during scipy tests. The PR was not updated for 4 years and in 2017 @rgommers closed the PR as MSVC + gfortan now worked and they continuous integration for it.
8454	This PR fixes #8389 which was about using single letter variable names that were ambiguous (I, O, l). This was an issue because certain letters look like numbers in a different font.	The problematic letters were replaced by different letters, for instance I was replaced by K. In the discussion it was pointed out that instead of renaming to a different letter, why not replace with longer meaningful variable names. The benefits did not justify the code churn and the PRs that were about fixing variable names and minor style issues were not encouraged by @pv. This PR was rejected.
7033	In this pull request, the author has changed the doc string of hyp2f1 function and included all the necessary details from the PR #6966 (Hypergeometric Functions documentation is lacking).	As a root cause analysis method, @person142 ran generate_ufuncs.py for these changes to actually show up in the published docs. However, it was later indetified that the author's issue was not with relevance to hyp2f1 and noted that the pattern was like eval_jacobi. It was identified that there was a misunderstanding that the author had in relevance to the conceptualization and later this PR was closed by the author upon seeking inputs from the other contributors in the channel as an ineffective and incorrect approach.

ID	Contributions	Decision
3893	In this pull request, the author has presented a special case of warning that was encountered "WARNING: document isn't included in any toctree" at base.rst and has noted that the default was overridden by custom orphan:1. This was also concurrently printed in all output files.	Although there were many questions raised on why the warning was to be removed, the author clearly defines that the warning was inappropriate and could lead to misleading error modifications in future. Hence, as a solution, the orphan settings were tracked down and changes in the 'doc-sphinx-orphan' were made by @rgommers. However, it was noted that the warnings generated in the autodoc weren't included in the fix and they were unable to work on the same. At the end, the author executed the same and figured out that with sphinx 1.2.2, python 2.7.8, and numpy 1.8.1 (but the numpydoc sphinx extension is from numpy's master branch), this change reduces the number of warnings from 406 to 323. Beyond this the author or the contributors could not work to reduce the warnings and hence they decided to close the PR as an unsuccessful attempt as the contribution was not sufficient. Finally, it was noted that the incorrect approach was used for this rectification, and they decided to close the PR as incorrect approach rather than insufficient output.

ID	Contributions	Decision
5226	In this pull request, the author has presented a special case of rewriting the Fotran code to compute the zeros of Ys in subroutine JYZO of scipy.special. The initial subroutine JYZO in specfun.f computes the zeros of J, J', Y and Y' using Newton's method. This is the function used by jn_zeros, jnp_zeros, yn_zeros and ynp_zeros. The problem was that the code that computed the zeros of Y' did not always provide a good initial guess for Newton's method. When the code saw that Newton's method had converged to a previously found zero of Y', it added pi to the initial guess and tried again. This did not always work, and it didn't prevent the code from converging to a zero that was larger than the expected zero. Neither Newton's method nor the retry loop were protected from iterating indefinitely. In the particular case of jn_zeros(281, 6), eventually it used an initial guess for a zero of Y' that put Newton's method into an infinite loop. As a solution, a fix was made to use the previously computed zeros of J as the initial guesses of the zeros of Y', and then only allow at most 10 Newton iterations. (After that, NAN is returned.) This works very well. I haven't found a case where Newton's method fails to converge.	Upon testing of the code, it was reported by the author that special.jn_zeros(281, 6) hangs and was referenced to PR #4690 for resolution. It was later figured out that for large values, there was an error message and code hangs were reported. However, there were no solutions provided to this problem and hence, the author decided to close the issue without any resolution mechanism. PR was closed due to no improvement and inactivity.

ID	Contributions	Decision
4379	In this pull request, the author has focussed on reductions that cannot be auto vectorized by gcc without adding unsafe-math optimization flags, so vectorize them manually for SSE2 which is available on all amd64 machines. This changes the numerical result as the summation ordering is different, but the change should be only in the order of a few machine precisions. Also, the author has requested for GIL computations releases. Also, the author has broken the code base for ease of testing. Upon code review, the contributors reported that option (+ -O3/-ftree-vectorize) gcc can vectorize but gcc specific could not be done and also it was not recommended for the semantic flags on a per function basis. Additionally, it was noted that the performance improves by a factor of 1.5x-2x for appropriate inputs. An RCA was conducted, and it was found that unrolling of the loop functions benefitted for GCC. However, the performance was improved only by 10% which was negligible as per the contributors.	When the Scipy team were involved in the conversation, it was noted that as per the policy they cannot speed by 1.5x-2x and hence they were unable to proceed with the request. The reason was the speeding up policy did not aid substantial results in the performance for them to consider it as a major enhancement level change. Hence the PR was closed unsuccessfully. However, the specifics of GCC optimization was carried over to PR #4551 while the auto vectorization was dropped with this PR. It was closed due to insufficient results to substantiate the effort of the auto vectorization – GCC.
3417	In this pull request, the author has referred to an extension work on constrained optimization, based on nonlin leastsq where he states that it would bring extra to scipy as slsqp seems present. To make it more visible what slsqp can do, it was suggested by adding an example also in the reference, and not only in the tutorial. Example taken here was based on the main section in slsqp.py and different from the tutorial which uses minimize directly.	Upon acting to the proposal, @coveralls noted that there were no changes when pulling 0f8416b on bmcage:slsqp into 3879d21 on scipy:master. Hence, a review was conducted by @dlax and it was reported that minimize (from the code propsed) did not have a disp keyword argument and _minimize_slsqp was private initialization. Also, it was identified and concluded that minimize to fmin_slsqp do not have any comparisons and hence had to be deprecated for fmin_slsqp. However, at the end, there were no resolutions steps for the request of the author and hence the author closed the PR in an unsuccessful attempt. It was closed as approach was not required or did not add value to the proposed problem of adding example to slsqp

22.13   Appendix B: Codification analysis

For codifying the tags we used are:

- Update: All comments that contain information about new changes to the code by the author of the PR is tagged as update.

- Conceptualization: All comments that ask for understanding of the code or the logic behind the approach are tagged as conceptualization. This includes the comments where they give suggestions or offer alternatives to approach and the author of PR explains his rationale.

- Testing: This includes all comments related to testing and testing results

- Misc: This includes random remarks and issues about github merge/rebase etc.

- Code change: This includes all syntax related comments and comments about suggestions to use different function.

- Performance: Comments about performance gain or regression are classified under performance tag

- Review: All reviews to the PR are tagged under review

- build: All comments about build issues or build process

- Dependency: All comments about dependencies and library errors etc are classified under this

- Configuration: Comments about setting up accounts and admin related matters come under this tag.

22.14   Appendix C: Technical Debt Glossary

1. **Maintainability**

In best case scenario, leaving maintainability-related issues means that even the best developers will have a hard time when they try to introduce new code changes. In the worst case scenario, developers may be confused by the state of the code, thereby introducing additional errors in the process of making code changes.

2. **Code Smells**

Code smells are usually not bugs—they are not technically incorrect and do not currently prevent the program from functioning. Instead, they indicate weaknesses in design that may be slowing down development or increasing the risk of bugs or failures in the future. Bad code smells can be an indicator of factors that contribute to technical debt.

3. **Maintainability Ratio**

The maintainability ratio is a metric given to a project related to the value of Technical Debt Ratio. For all technical debt ration values less than 5%, a rating of A was provided.

4. **Measures of reducing Code complexity thereby improving Pylint rating**

- make sure your code writing style is conform to what Pylint is expecting (or tune Pylint to match your style / conventions). This includes function, variables, class, method names, spaces at various places, etc.

- write Python code in an as static as convenient way, and avoid dynamic tricks.
- write doc-strings

5. **Reliability** In static analysis, the parser identifies specific methods and isolates them, emulating their behaviour when ported into other software modules.

Once isolated, the algorithm will generate arguments and read method outputs, expecting the outputs to contain specific information and formatting. It will raise flags for any anomalies in a method.

6. **Security**
   The primary cause of software vulnerabilities are defects, bugs and logic flaws. Organisations should take proactive actions to decrease vulnerabilities before software deployment. Hence educating the employees on secure coding techniques can help eliminate these vulnerabilities.

7. **Code Coverage (specifications)** In specific, it is a measure of how many lines/blocks/arcs of code are executed while the automated tests are running.

22.15   Appendix D

D1. Resolving Testing Debt

Testing Debt can occur owing to several kinds of testing methods such as: 1. integration testing 2. system testing 3. security testing 4. usability testing 5. performance testing 6. unit testing 7. While these tests help in identifying accurate test results for a software in pre-release phase, often the associated debt is ignored.

Here, we have highlighted some important methods which maybe incorporated for SciPy.

Additionally, these guidelines maybe useful for other large repository project like SciPy too. They are: 1. Having a clear augmented testing platform for SciPy owing to multiple language dependencies in the project (from codebase reference) 2. Tesitng the development versions on a regular basis for SciPy and not at the time of pre-release 3. Since, testing debt cannot completely be erradicated, we need to prioritize the best resolutions and work for SciPy, which is not presently being done. With these, we believe that the testing debt maybe reduced for an instance like SciPy.

D2. Contributions

- **Issue URL:** https://github.com/scipy/scipy/issues/10014
- **Pull-Request URL:** https://github.com/scipy/scipy/pull/10053
- **Description**: The existing code base lacked an intuitive description of weibull_max_gen and weibull_min_gen classes in file scipy > stats > `_continous_distns.py`. We added a description to both of these classes based on the issue and a brief research on wikipedia.

Chapter 23

Servo

23.1 The Parallel Browser Engine Project

By Dominique van Cuilenborg, Bart van Schaick, Fabian Stelmach and Aron Zwaan

Servo is a modern, high-performance browser engine. It takes advantage of the memory and concurrency management provided by the Rust language. The project was started by Mozilla Research and is being built by a community of individual contributors from companies such as Mozilla and Samsung. The long-term goal of Servo is to incrementally replace components in Firefox. Other than that, Servo is also trying to become a stand-alone browser.

In this chapter, we provide a detailed insight into the architecture and development of Servo. We start this chapter by analyzing the development process of Servo, this involves analyzing pull requests, the stakeholders and the context. We then continue with the architecture and conclude with a section on technical debt.

23.2 Table of Contents

Figure 23.1: Servo Logo

23.2.1 Contact

If you are interested in the Servo project and would like to know more, we would recommend you to contact @jdm. He has left us with very helpful feedback on our Pull Requests and is very active in the Servo community.

23.3 Pull Requests

A total of 20 pull requests have been analyzed in order to identify patterns that differentiate successful pull requests from unsuccessful ones. The pull requests with the highest number of comments have been selected. The list of analyzed pull requests is present in appendix 1. In this section, we aim to provide a summary of the decision-making process for the acceptance of pull requests.

23.3.1 Commonly Occurring Patterns in Pull Requests

The most commonly occurring discussion point is the automated testing suite. The testing suite can be automatically ran on a pull request after one of the core team members has given permission to do so. Successfully running the test suite on a particular pull request is required in order to successfully merge a pull request. This requirement is often difficult to achieve, since pull requests with new features or bugfixes are required to add relevant tests and the testing suite currently runs on 25 different targets, including Windows, Linux, MacOS and Android. Quite often, developers have to change their code in order to pass the tests on all the platforms. Most discussions revolve around how the failing tests can be fixed.

The second most occurring discussion point is general code quality. The Servo team welcomes contributions from non-experienced programmers, and therefore, it is often needed for the experienced members to request adjustments to the pull requests. These adjustments can consist of re-structuring of the code as well as

re-implementing some functionality in a more efficient way. Noticeably, apart from the pull request checklist, the only other official guideline is the set of strict rules for styling Rust code by the Rust language team. No further official guidelines are officially presented by the Servo team.

23.3.2 Commonly Occurring Patterns in Unsuccessful Pull Requests

Unsuccessful pull requests differ from successful pull requests in that they are not merged into the main branch of Servo. After thorough research, we have found that most of the rejected pull requests were closed simply due to inactivity of the creator of the pull request. Many pull requests are dropped in the stage where the failing tests need to be fixed. This often proves to be more of a challenge than the creator originally expected. As such, these pull requests are simply left untouched and closed after some time of inactivity.

The second most commonly occurring reason for refusal of a pull request is the supersedence by another pull request. This can happen when a developer has written a solution to address some issue, but that solution is only partial, and so it does not fully solve the problem or does not successfully pass the test suite. In this case, another developer may decide to use this partial solution and complete it in another pull request. In the case that the completed solution is merged, the original pull request is automatically rejected.

23.3.3 Decision process

The decision process of the acceptance of the pull request by the Servo team depends mostly on the quality of the supplied code. If the solution supplied is of acceptable quality and it passes the testing suite, then the pull request is generally accepted.

The pull requests are rarely challenged for the actual functionality they implement. This is possible due to the fact that most of the pull requests are targeted to fix a particular issue that has been identified by the developers beforehand. Issues which do not need fixing, get shut down before a pull request for them is made.

23.3.4 Integrators

The integrators, those who approve or rejects these PRs, are found in the Governance. This governance states the core group of Servo (@larsbergstrom, @metajack, @jdm and @pcwalton). This core group can be seen as the integrators of the Servo project, since they govern the direction of Servo and make all the final decisions.

23.4 Stakeholders

For a research browser-engine project, it is reasonable that many stakeholders are involved. Be it companies, who of course have an interest in the product itself, or simply the developers working on Servo. We compiled a list of stakeholders using Rozanski and Woods classification. More on this classification can be found in their book Software Systems Architecture.

23.4.1 Rozanski and Woods classification

23.4.1.1 Acquirers

The primary stakeholder is Mozilla and Servo can be seen as a playground for Mozilla's Firefox browser. As stated on the roadmap, if certain functionality gets implemented in Servo, Mozilla may merge this with their main Firefox browser. Most of the roadmap is set out by the Mozilla employee @jdm, this ensures that the project takes the direction Mozilla wants.

23.4.1.2 Assessors

When it comes to conformance to standards, a browser has the comply with various browser standards, such as those stated by W3C. Ensuring this is the responsibility of the core developer team, where various developers have their own specialization. For example, @nox takes care to make sure that the behavior of the CSS renderer in Servo complies to the browser standards. Furthermore, regarding legal regulations, the Servo project uses Mozilla Public License 2.0 (MPL). This license clearly states that the contributors are not liable and there are no warranties for using Servo.

23.4.1.3 Communicators

The communicators are the core servo team. They wrote several wiki pages on the design of Servo to ensure that every stakeholder can get up-to-date with the architecture of the program. Other than the design, there are also numerous wiki pages written on the development guidelines of Servo. These pages ensure that new developers know what is expected of them.

23.4.1.4 Developers

The developers are the contributors on Github. Most, but not all, of the main developers work for Mozilla. This once again strengthens Mozilla's stakeholder position in the development of Servo. Other than these main developers there are still lots of Rust enthusiasts committing to the project. The developers are involved to actually implement the ideas of the other stakeholders.

23.4.1.5 Maintainers

Servo is not yet ready for full deployment; however, certain subsystems certainly are. These subsystems are maintained by the group of core developers, which consists mostly of Mozilla employees. They are involved to make sure that the project is well documented and limit technical debt.

23.4.1.6 Suppliers

The main sponsor of Servo is Mozilla, they provided the necessary resources to get the project to where it is today. One of the resources provided by Mozilla are the employees actively working on this project.

Furthermore, Mozilla also provides the build server and many other Rust development tools to streamline the development of Servo.

23.4.1.7 Support Staff

Since the product is not meant to be your daily driver when it comes to browsing, there is no real support staff yet. Since this project is targeted at developers, the main support staff can be considered as the developers on GitHub, which you can contact using the issue tracker.

23.4.1.8 Testers

The developers themselves are responsible for testing their own commits. Apart from that fact that they should choose a test approach that matches the altered component, which can be found in CONTRIBUT-ING.md, no strict rules are enforced.

23.4.1.9 Users

There are no real users for this project. Rather than the final product being the goal, the goal is to function as a test-bench for Mozilla's Firefox browser. Therefore, Mozilla can be seen as the main user. While Servo as a whole is not used by people; certain subsystems, such as WebRender, are already used to create applications. An example of such an application is Azul which allows for web applications to run on your desktop, similarly to Electron.

23.4.2 Other Stakeholders

Other than the classification of Rozanski and Woods, other stakeholders can be found.

23.4.2.1 Researchers

The browser is a playground for Researchers, since servo is mostly meant as a research project and not yet as a deployable product. Servo has been used to write many papers regarding the parallelism of browsers, improving browser performance and many other topics.

23.4.2.2 Competitors

Competitors could also be seen as stakeholders of Servo, since the success (or failure) of the project may result in a change of the competitors their market position. The main competitors of Servo are Chromium and WebKit.

23.5   Context View

The context view shows the relationships, dependencies and interactions between Servo and its environment. This context view allows us to easily see the connections between the various external entities and Servo.
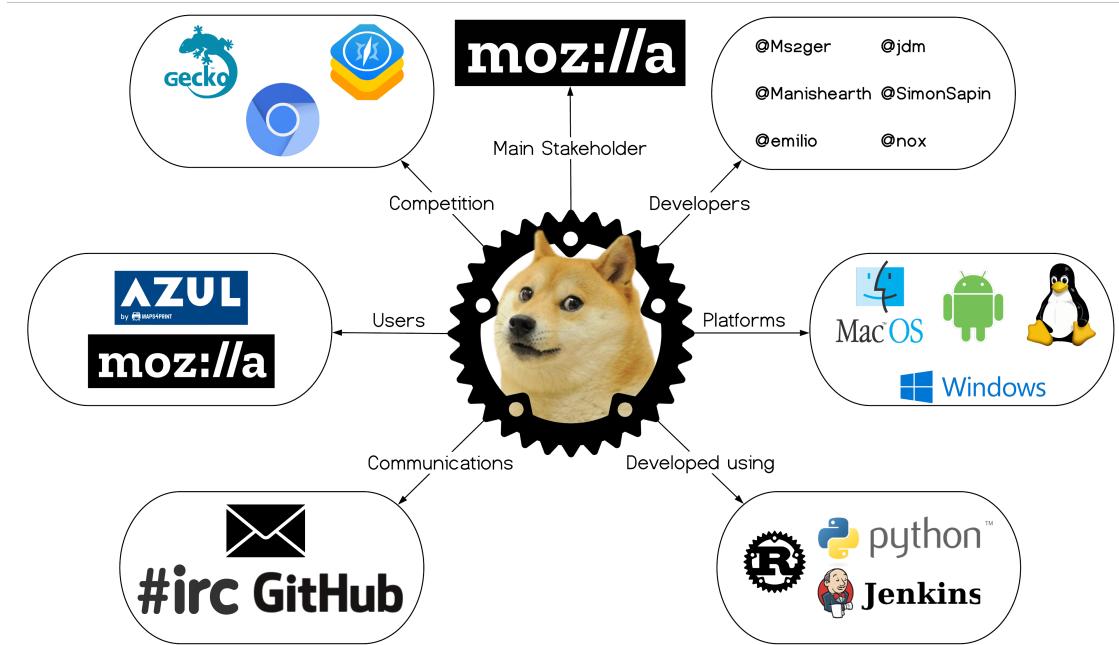
Figure 23.2: Visualization of the context view

Using this visualization we can analyze the context. Servo is written in Rust, a programming language developed by Mozilla. Furthermore, Jenkins and Python are used to allow for a streamlined development process. The source code and issue management are performed on GitHub, the source code and all issues are kept here. Other than GitHub, communication is done on the Mozilla's IRC (irc.mozilla.org) and the Servo mailing list.

The main competitors of Servo are all other browser engines. These engines are usually made by very big companies (Microsoft, Apple, Mozilla) with lots of funding. However, it is interesting to note that Gecko (Mozilla) and Servo are not real competitors. Their relationship is mutualistic, since Mozilla supports the development of Servo and Servo functions as a test bench for Mozilla's Gecko engine.

Servo is meant to service all major platforms. It did not support Windows and iOS during the first few years of development, due to technical difficulties that come with having a browser engine work cross-platform. However, as the project got a bigger development team, Windows and iOS support was added.

It is interesting to see that there are not that many people using Servo, especially considering the amount of development that goes into Servo and the competitors it is fighting up against. However, due to the mutualism between Gecko and Servo, there is still a very important reason to continue development.

23.6   Development View

The development view supports the design, build and testing of Servo, by describing the architecture that supports the software development process. Code structure and module organization are important aspects of this architecture, which also involves identifying areas of common processing, codeline organization and standardization of design and testing approaches. This chapter addresses these concerns by depicting a module structure, common design and codeline model for Servo.

23.6.1   Module Structure Model

Servo's logic is divided into a number of components. At the moment Servo contains 52 components, each with its own internal and external dependencies. Modeling these components into one giant diagram would be cumbersome, time-consuming and above all, not aid developers in getting a better understanding of the system. Instead a high-level overview is given that will explain the relations between the most important components and their interactions with the outside world.

Servo is advertised as the 'Parallel Browser Engine Project'. This leaves very little to the imagination of the intention of the developers. To aid this goal, a task-based architecture is adopted in which major components are loosely coupled and modeled as threads. These components can interact with each other by passing messages over channels or by borrowing (sharing a reference to) a data structure.

In Servo, the thread that owns these channels is called the **constellation**. The constellation can be thought of as a tab in a browser (Servo currently supports only a single tab). It tracks all of the information related to the current tab and updates its state based on the messages it receives from other components. The constellation also manages all pipeline objects. A pipeline encapsulates a means of communication with the script, layout and render thread. Each pipeline is responsible for correctly displaying a particular document in its assigned window or frame.

Apart from the constellation, some other major components of Servo are identified and explained in greater detail below.

- The **Script** thread creates and stores the DOM representation of the document, by invoking the HTML Parser and executing Javascript code.
- The **Layout** thread takes a snapshot of the DOM provided by the script thread applies styles and calculates the layout of nodes. From this layout, it builds a display list, which is basically a list of concrete rendering instructions.
- The **Render** thread is tasked with translating the display list it got from the layout thread into a set of drawing commands.
- The **Compositor** is charged with two tasks. The first is compositing the views of several render threads and display this on screen. Furthermore, the compositor receives input events from the operating system and should forward these to the constellation.
- The **Net** or network thread handles all network interactions, such as creating HTTP requests and fetching data.
- **Data** is not identifiable in the way it is represented in the diagram, but should be seen as a combination of resource, storage and caching components that aid the data persistence department of the browser.
- The Servo **Media** component is tasked with the playback of numerous multimedia file formats. It is developed as a standalone project.
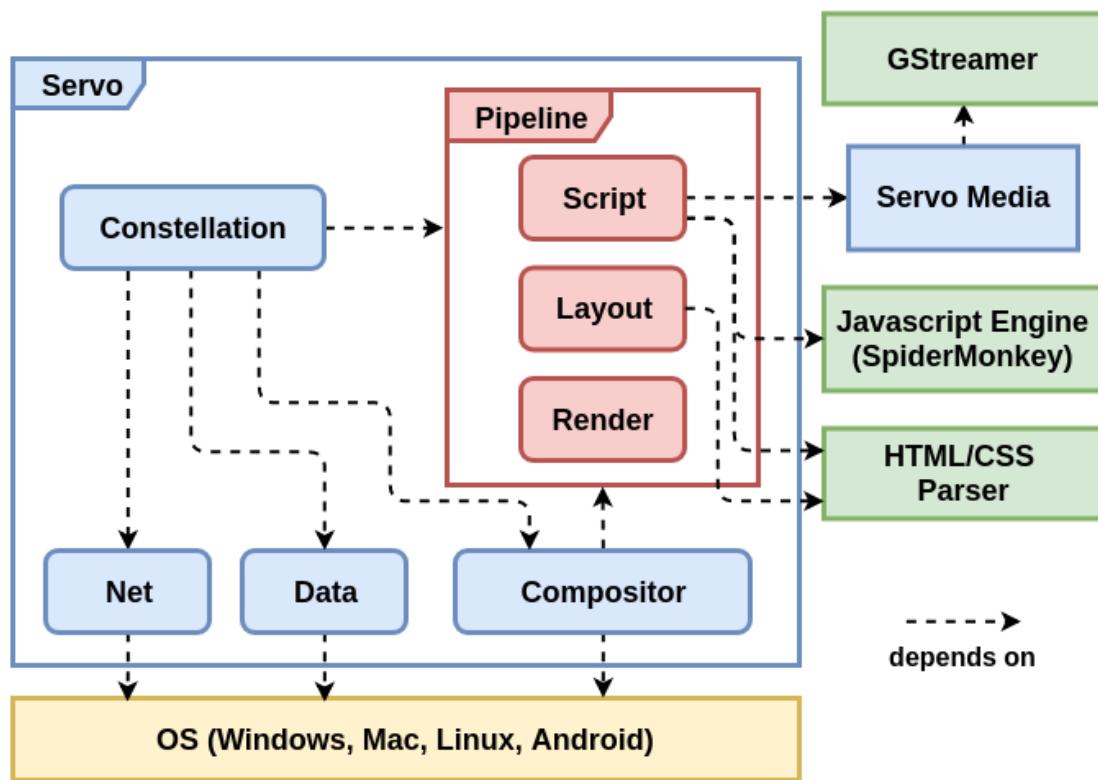
Figure 23.3: Module Structure Model

23.6.2   Common Design Model

To maximize commonality, reduce risk and duplication of effort on implementation on the component, module and function level, a set of design constraints is used. These design constraints identify standard approaches to be used when solving certain types of problems. Apart from some standard coding rules, Servo does not explicitly mention design approaches in its documentation. However, areas of common processing, standard design approaches and common software used across all components can certainly be identified.

23.6.2.1   Common Processing

In a highly concurrent and parallel system such as Servo, it is of utmost importance to specify a set of clear rules regarding the responsibility for and communication between threads. They specify how threads should act to avoid or recover from failure.

In the case of Servo, most threads can and are allowed to fail when encountering certain errors. The constellation is not, as it is unique in its kind and tasked with the crash reporting of other threads. The Concurrency View explains the threads and their relationships in great detail, among measures to prevent them from failure.

23.6.2.2   Standard Design Approach

Servo follows Rust's guidelines on file and directory structure. Servo's so-called components are designed like packages, better known as crates in Rust. A component has the following directory structure:

```
- component-x/
  |- bar/ (optional 0-n)
  |- Cargo.toml
  |- lib.rs
  |- build.rs (optional 1)
  |- foo.rs (optional 0-m)
```

A module always contains a `Cargo.toml` and `lib.rs` file. The `Cargo.toml` file is called the manifest and contains general information such as the name and version of the package, but also lists dependencies on other packages. The `lib.rs` file declares the modules (foo and bar in the example). `build.rs` is used to build third-party non-Rust code before the compilation of anything else in the package.

23.6.2.3   Common Software

Servo uses Rust's standard logging crate log through its codebase. It contains several logging levels, of which *error*, *warn*, *info* and *debug* are the ones used. A short message is added to describe the event that occurred. The constellation tracks any *error* and *warn* messages generated by itself and other components.

23.6.3 Codeline Model

In a giant open source project like Servo, it is important to have a clear and detailed specification of the organization of the system's code. A good codeline model incorporates a definition on the overall structure of the code and the automated tools used to build, test, release and deploy the software.

23.6.3.1 Code Structure

For developers who just started working on the project, it is important to know where certain source files and tools are located in the project structure, as this will speed up the development process significantly. In the next table, an overview of Servo's directory structure is presented, including a description of what developers can expect to find in these directories.

Directory	Description
components	Basically the source files, a directory containing all Servo's components
docs	Documentation useful to new developers, ranging from debugging tips to the style developers should adhere to
etc	A number of useful tools and scripts for developers
ports	Code to create a Servo instance
python	Several Python modules to support Servo development
resources	Resources needed at runtime
support	Code needed for Servo to run on different platforms
tests	Files and tools for testing

23.6.3.2 Build, Integration and Test Approach

Servo is hosted on Github, which gives developers the option to open pull requests and issues and have discussions on them. The project welcomes contributions from everyone, in the form of a pull request, which is reviewed by at least one of the core contributors. Pull requests must meet a number of requirements. First off, it should fix an issue claimed by the contributor and is expected to add or alter relevant tests. Secondly, commits should be as small as possible and build successfully, independent of other commits. At last, all coding guidelines should be followed.

Mozilla's Mach tool is provided to orchestrate the build and other tasks. It provides several self-explanatory options for building and testing of the software, making the developer's life easier.

When new functionality is pushed to Servo's repository the code is built and tested using Travis CI, AppVeyor and TaskCluster. Travis CI and AppVeyor run tests on Linux and Windows, respectively, while TaskCluster runs certain commands like a formatter. This is not enough to assure that a pull request will not break things after being merged. Another pull request may be merged in the meantime. Therefore Servo uses Homu, which runs CI again, and only merges when everything succeeds.

23.7   Concurrency View

Servo advertises itself explicitly as a *parallel* browser engine. This implied concurrency drives the design of the project. Therefore, in order to understand the project's architecture, it is very helpful to investigate how concurrency is dealt with in Servo. Therefore we elaborate more on this topic in this section.

First, we will describe what different thread types servo has, and how thread communication works. After that, we will investigate messages passing patterns some deeper.

23.7.1   Process Model

Servo has a task-based architecture, in which every major component executes in its own thread or process. Different threads communicate through channels. The full Process model of Servo can be seen in the diagram below.
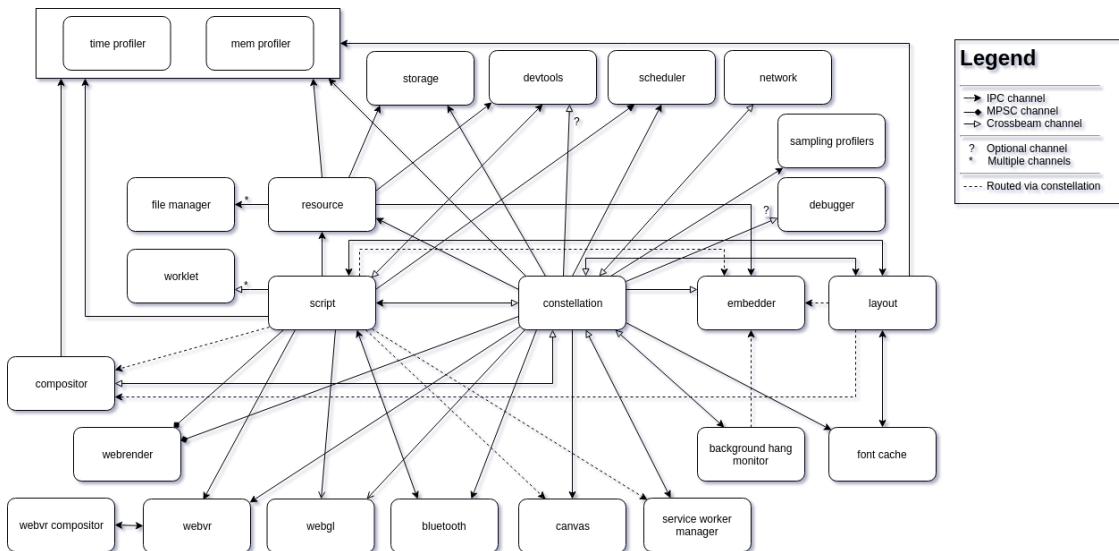
Figure 23.4: Visualization of the Process Model

The Servo project uses three different message passing channel implementations, represented by the three different arrows in the figure. These implementations are explained in the subsequent section. The process model is clearly centered around the constellation struct, as seen in the figure. The two important roles of this struct are the maintenance of channels to threads and passing messages between those threads. Inter-thread communication can, however, also occur directly between threads, without passing through the constellation. Finally, some channels, such as the debugger and devtools, are optional.

23.7.1.1   Communication

Servo uses three different channel implementations:

- MPSC: The Multi-Producer, Single-Consumer channel implementation from the Rust standard library. This implementation is only used to communicate with the webrender component.
- Crossbeam-channel: This channel implementation is used throughout the application. It has some benefits over MPSC. For example, it can block on multiple receivers using the `select!` macro. This feature is used in the constellation, background hang monitor, layout thread, script thread and service worker manager. Moreover, its receivers can be cloned, effectively making this an MPMC (Multiple Producer, Multiple Consumer) channel. Contrary to MPSC, in this implementation, sending is blocking the thread.
- IPC-channel: This is a custom channel implementation specifically for Servo. It supports inter-thread communication as well as inter-process communication. This makes it especially suitable for Servo, since it is optionally multi-process (specified by the `-M` switch). Like crossbeam-channel, the send operation might be blocking.

23.7.1.2 Deadlock Prevention

Probably, the most well-known problem in concurrent applications is deadlock. Deadlock occurs when different threads are mutually waiting for each other to make progress, resulting in a situation where no single thread can make progress, so that the process got stuck. Servo mitigates the risk deadlock by avoiding the circular wait condition. To do so, it defines a hierarchy between threads. This hierarchy is only defined in documentation, but not enforced by a compile or runtime error.

The primary source of a thread waiting for another thread are blocking sends. This risk could be avoided by only using MPSC channels, but then Servo could not take advantage of the other channel implementations. Instead, channels are routed into other channels, using the ROUTER struct from IPC-channel. This router uses a dedicated thread to forward messages, and therefore sends become non-blocking. We identified three different patterns in which message routing is used:

- Routing IPC receivers to MPSC senders: This enables non-blocking inter-process sends.
- Routing IPC receivers to crossbeam receivers: This enables the receiver thread to try to read from multiple receivers using the `select!` macro.
- Routing IPC receivers to listener objects, that process messages. This pattern will be further explained in the next subsection.

23.7.1.3 Data Races

Another commonly occurring problem in concurrent applications is the presence of data races. Servo does not explicitly deal with data races, since by design, safe Rust guarantees the absence of data races. This is guaranteed by the strong ownership system of the language, which prevents aliasing of mutable references, thus making data races impossible. Note that this only holds for data races and not general race conditions. The latter is addressed manually by contributors, using synchronization. This solution is viable, but due to its complexity, it often manifests itself in bugs [1] [2] [3].

23.7.2 Passed Messages

Usually, a concurrency viewpoint contains a state model, which "describes the set of states that runtime elements can be in and the valid transitions between those states". However, since all the thread interactions

in Servo are relatively independent, and there are several hundreds of different message types, such a model would be cumbersome to make, and not very helpful. Instead of such a model, we describe the general approaches Servo uses to update the state of individual components.

In some situations, the sending thread does not care what the result of a message is. In a state model, this would indicate no state transition on the sending thread. This is implemented by just performing a (non-blocking) send, and resuming execution.

23.7.2.1   Task Queue

In some situations, tasks are queued in the channel, to be processed later. This can be the case when the receiving thread might be busy. In this case, the sending thread adds the message to a queue, so that the receiving thread can process them when it becomes available again.

This pattern can easily be spotted in the script thread, where all pending events are stored in a vector, and processed together. A similar pattern is employed in the worker event loop.

23.7.2.2   Send Reply Channel

A special type of callbacks occurs when a message passes a sender along, which the receiver can use to reply to that message. This pattern occurs many times, for example when the constellation requests the current epoch from the layout thread. An advantage of this pattern is that the sender can block on the passed channel, achieving some sort of synchronous communication.

23.8   Technical Debt

In this section, we will assess the technical debt of the system, using the SOLID principles. We used these principles since they are well-known, well-defined and apply to the Rust language and the target project. However, we skipped the Liskov Substitution Principle, since Rust does not support inheritance. We verified all properties at module-level, since that is the most basic coherent unit in Rust. After analyzing the SOLID principles, we analyze the testing debt, how developers manage their technical debt, and how the technical debt evolved over time. Finally, we give an overall assessment of this project's technical debt.

Analyzing the architecture was quite tedious, since there are no good automated tools available to do so in Rust. The only serious candidate was clippy (an extensive linter), but running this analysis tool failed on the project, since it is not incorporated in the Mach build system. Therefore, the SOLID analysis was done manually. Therefore, the figures we gathered might be somewhat inaccurate, since the name resolution in IntelliJ is not fully implemented, and Rust does not require explicit types everywhere. Therefore, given figures will, in general, be underestimations.

23.8.1   Single Responsibility Principle

Regarding this principle, we think it is adequately taken care of, although there is still some room for improvement.

We could find several violations of this principle:

Module

Issues

`script/dom/window.rs`

Functions `base64_btoa` and `base64_atob` do not use any of `Windows` properties, and therefore should be in a separate utility module.

Functions `cancel_all_tasks` and `cancel_all_tasks_from_source` should be in `script/task_manager.rs`, because they only manipulate tasks in the task manager.

`layout/block.rs`

Function `assign_block_size_block_base` is 372 lines long, and local variable usage analysis shows it can be split with some effort.

23.8.2 Open Closed Principle

This pattern is difficult to enforce fully in Rust, since idiomatic Rust uses matching on enums a lot. Therefore changing an enum often requires fixing several match statements (as we did in our first PR). The advantage that Rust match statements have compared to switch statements in C-like languages is that the compiler warns if not all possible patterns are matched.

Moreover, throughout the project, struct properties are hidden and accessed using getters and setters consequently.

23.8.3 Interface Segregation Principle

In Rust, the construct that matches the concept of an interface in OOP languages is a trait. In servo, traits are usually kept small, one to three functions, with some exceptions. For example, the `LayoutRPC` trait has 11 functions, which are never used together in a function, although all are used in different functions in `window.rs`. Therefore, technically, the interface could be split, although that would not improve the understandability of the code. Other occurrences of this problem are `ThreadSafeLayoutNode` (31 functions, of which 6 are used together once), `ThreadSafeLayoutElement` (15 functions, never referenced directly), `TNode`(17 functions, 4 used together) and `TElement` (71 functions, 4 used together).

23.8.4 Dependency Inversion Principle

This pattern is implemented really well in Servo. Many components consist of a module defining the traits and data types, and a module containing the implementation. Other modules usually reference the module defining the traits. For example, there exists a module `bluetooth` and a module `bluetooth_traits`, where the first implements the latter. Now there is one module having a dependency on `bluetooth`, while there are 5 modules referencing `bluetooth_traits`. The same pattern happens in `script` (2 references) and `script_traits` (12 references), and several other modules.

23.8.5   Testing Debt

The testing approach taken is pretty sophisticated for a Rust project. There are three categories of tests, which we discuss in detail in the following subsections. Unfortunately, it is not possible to measure coverage, since the rust coverage tools are not compatible with the mach build system, which servo uses, and the latter 2 test approaches.

23.8.5.1   Unit Tests

Firstly, there are some unit tests, testing important components of the project, like the scripting engine and the styling. However, there are 73 dedicated test files, 6 files containing source code and tests (which is quite usual in Rust), containing a total of 13k lines of code (measured using tokei). For comparison: the project has almost 1000 source code files (excluding generated code) with almost 250k lines of code. This suggests a very strong lack of unit testing. We think this is serious technical debt, since it makes tracing back issues quite hard, and leaves a lot of uncertainty for components that implement behavior that is not directly related to the following test categories.

23.8.5.2   Web Platform Tests

Web platform tests(WPT) is "a W3C-coordinated attempt to build a cross-browser test suite for the Web-platform stack". This is a massive test suite tests, testing a browser against the official W3C and WHATWG specifications. Servo uses a bot to keep the WPT tests updated. However, examining the configuration file determining what tests are included, we see the vast majority of tests is skipped. This is most likely due to the fact Servo is not compatible with the full specification yet. Nevertheless, 25014 tests from the suite are run, which guards for regressions.

23.8.5.3   CSS Tests

Like the web platform tests, the W3C established a set of CSS tests as well, which are partially included in the Servo test suite. The testing guide states `./mach test-css` would execute those. However, trying this failed. We suspect that the CSS tests are included in the command to run the wpt tests, which would imply that this is outdated information.

23.8.5.4   Continuous Integration

The unit tests, together with the static analysis, are run in CI on any PR for Linux as well as for Windows. Before a PR is merged, the build bot runs the full test suite on 25 different targets, ranging from Windows to different variants of Linux, MacOS, Android, ARM and Magic-leap.

On top of the integration testing, a bot is employed that warns if code is touched, but no tests were changed (see this PR for an example). This could be a very useful means to maintain a good test suite. However, as we saw in the section on Unit Tests, there is a lack of unit tests. Therefore, we think it is not used that well. In our opinion, it would be possible to monitor the testing effort better by employing a CI check that validates if some coverage criterion is not dropped.

23.8.6 Discussion

The developers do monitor the technical debt of their project. There are approximately 1350 TODO/FIXME comments in the code. The repository has 315 issues labeled `I-cleanup`, of which 28 are still open, and 322 issues labeled `I-refactor`, of which 41 are still open. This suggests there is a considerable amount of work that can be done to reduce technical debt. Moreover, there are much more comments tracking technical debt than issues monitoring it, which suggests that some issues are out of sight. This could be solved by introducing a static analysis tool in the CI that checks the diff of the PR for TODO/FIXME comments, and blocks merging until they are gone, or at least suggests to make issues for them.

23.8.7 Evolution
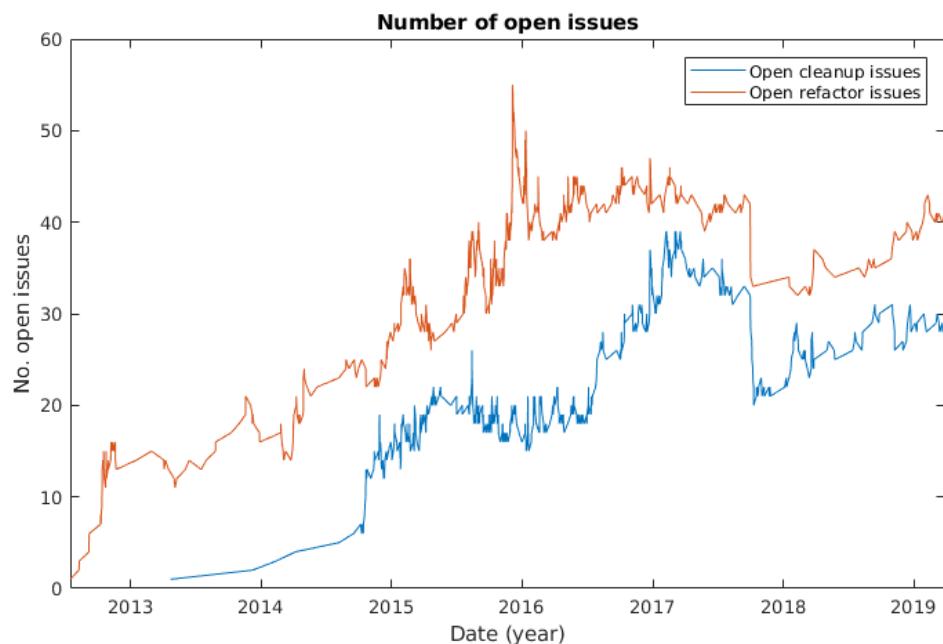
Figure 23.5: Open Refactor Issues

The level of technical debt is generally increasing slowly, as can be seen in the number of open Issues plot. This is not strange if we consider the growing project size. We could not observe bursts of closed `I-cleanup` and `I-refactor` issues, except for the first week of October 2017. Analyzing the issues closed in that period learns that many outdated issues were closed back then. Some of these (like issue #4786) were more than 2 years old, indicating that they were not monitored closely.

23.8.8   Debt Summary

We can conclude that the code itself well organized albeit some exceptions. In addition to that, the CI infrastructure is outstanding, with test ran on many different targets before a change can be merged. However, we think the unit test suite is not adequate for the size, maturity and complexity of the project. The end-to-end tests will cover the most important behavior, but are not suitable to verify the internal workings of the browser engine. Finally, the technical debt, of which the developers are aware, is managed adequately.

We think the impact of the technical debt is moderate. Many regressions will be detected by the WPT tests, but the actual cause will be hard to trace down. With a proper set of unit tests, tracing down regressions is a lot easier. Moreover, since this is a playground/research project, no great interests are at stake when the project malfunctions.

23.9   Conclusion

In this chapter, we conclude that there are various stakeholders, the main one being Mozilla, and that the project does not have a lot of users for the scale of the project. This lack of users is mostly caused by the fact that Servo is meant as a research project and not for daily use.

We analyzed the general modularity of the project. We found that the project is heavily modularized, as is the goal of Servo, and that these modules are very loosely coupled. All these modules run on their own thread and communicate by passing messages. Furthermore, we analyzed the technical debt. We conclude that the project is fairly well organized; however, it should be said there is quite some technical debt regarding tests and open refactoring issues.

The inspection of the message-passing based concurrency model of Servo has revealed that the project has a carefully planned hierarchical architecture to prevent deadlocks. The use of the safe Rust language prevents data races from occurring. Finally, commonly occurring design patterns such as task queue and reply channel sending lower the complexity of the concurrency model, making it fearless to use.

All in all, we conclude that the architecture of servo is well organized. We think that there are still some improvements to be made, especially regarding technical debt, but in general, we think Servo is a great example of a well-engineered software architecture.

23.10   Appendices

23.10.1   Appendix 1 - Analyzed Pull Requests

A table of pull request used in the analysis in the chapter "Pull Requests".

Pull request ID	Title	Number of comments	Merged
21029	Upgrade to SM 60	433	Yes
21325	Replace mpsc with crossbeam-channel	249	Yes

Pull request ID	Title	Number of comments	Merged
8641	No more headless compositor. Just the normal one.	232	Yes
12186	Implement video-metadata check	190	Yes
5652	Kicking off a WebGL implementation	185	Yes
16176	Halve number of processes for test runs.	160	Yes
20755	Implement unhandledrejection event	220	Yes
10373	Enable WebGL tests	146	Yes
16508	Properly set origin of fetch requests	173	Yes
20678	Implement Window.open and related infrastructure	194	Yes
12989	Add headless rendering mode, and run WPT tests with Webrender enabled.	180	No
15852	Update to cargo-0.18.0-nightly (fa1b12a 2017-02-07)	97	No
20850	Investigate websocket timeouts	92	No
9410	Adding sync method to update atrr from inline style updates	105	No
11969	Use a winres to give servo.exe an icon on Windows	101	No
14764	[WIP] Make subsequent about:blank loads async	114	No
10604	Implement XMLHttpRequest.send(Document)	91	No
8374	CSS test fonts	71	No
11739	constellation: Don't ignore inconsistent frame-tree states when determining when to take a screenshot.	66	No

Pull request ID	Title	Number of comments	Merged
12858	Enable wpt WebGL tests on Linux	71	No

23.10.2   Appendix 2 - Contributions

Other than analyzing Servo, we also got a few of our pull requests merged into Servo. The following issues were picked up and fixed by our team:

- #22986 - HTMLIframeElement.set_visible is unused - fixed in #23007
- #22982 - Promise::new crashes if no compartment has been entered - fixed in #23158

This last PR has gotten a follow-up issue #23167, one of our team members is planning to follow up on this issue after this project is finished.

Chapter 24

Spring Boot - production-grade Spring-based Applications that you can "just run"

by Andrei Simion-Constantinescu (top right), Hendrig Sellik (top left), Milko Mitropolitsky (bottom left),

Viktoriya Kutsarova (bottom right)

24.1 Abstract

Spring Boot is an open source project part of the Java Spring Framework designed to simplify the creation of stand-alone, production-grade Spring-based applications. Spring Boot has revolutionized the way production-ready applications are developed, allowing developers to focus more on the application logic rather than spending time on boilerplate code to handle the necessary configurations and dependencies to run the application. Spring Boot is used in many enterprise solutions (e.g Netflix) for the fast building of massive applications, especially web-based ones. This chapter will be going from introducing Spring Boot's stakeholders and their roles in the decision making process to looking at the project from multiple perspectives like the Context View, Development View and Functional View. Finally, a research of the impact of Technical Debt is provided followed by the final conclusions of this project's complex architectural analysis.

24.2 Table of Contents

24.3   Introduction

Spring Framework for Java was born in February 2003 after Rod Johnson's book Expert One-on-One J2EE Design and Development [1] was published. In his book, the author described how a scalable high-quality application can be developed without Enterprise JavaBeans (EJB) framework using dependency injection in addition to ordinary java classes. Over the years, one of the most criticized aspects of the Spring Framework was the complex dependency management handled by massive XML configuration files.

Spring Boot was launched in April 2014 resolving the problem of complex XML configurations by having three notable features that make it both unique and easy to use:

- **intelligent auto-configuration** - setting the application based on the surrounding environment and information provided by the developer

- **stand-alone** - eliminating the need of deploying to a web server or any other special environment by embedding Tomcat, Jetty or Undertow servers directly

- **opinionated** - saving developer's time by configuring the most popular libraries by default in the most standard way

Due to the power of creating Spring-based applications that can "just run," Spring Boot's popularity grew faster. Spring Boot is being used by many large companies such as Netflix, American Express, ESPN, Mobile.de, Wix.com etc. To present a complete architectural description of Spring Boot, we follow the approach defined by Rozanski & Woods (2012) [2], starting with the Stakeholders analysis, followed by multiple architectural views such as the Context View, the Development View, the Functional View and continue with the Technical Debt analysis before drawing the final conclusions of this chapter.

24.4   Stakeholders

The stakeholders of Spring Boot are described in *Table 1*. Categories used correspond to the stakeholder types described by Rozanski & Woods (2012) [3]. In the Other stakeholders section, we have identified additional people and organizations concerned with the Spring Boot project.

Category	Stakeholders
Acquirers	Pivotal company, with their Board of Directors and leadership team.
Assessors	The Board of Directors of Pivotal have created a corporate governance framework.

---

[1]Johnson, Rod, and Juergen Hoeller. "Expert One-on-one J2EE Development without EJB." (2003).

[2]Rozanski, Nick, and Eóin Woods. Software systems architecture: working with stakeholders using viewpoints and perspectives. Addison-Wesley, 2011.

[3]Rozanski, Nick, and Eóin Woods. Software systems architecture: working with stakeholders using viewpoints and perspectives. Addison-Wesley, 2011.

Communicators	Pivotal offers Spring Boot training courses for developers, architects, system administrators, and technical managers. The official Spring Boot documentation and reference guide are created and maintained mainly by the core engineering team. There is also large community support on Gitter.
Developers	Main developers of the system are part of the Pivotal company. Most active ones are Stéphane Nicoll, Andy Wilkinson, Madhura Bhave. The founder Dave Syer and co-founder Phil Webb have also contributed a lot in the development process.
Maintainers	Overlapping with developers. Mostly Stéphane Nicoll, Andy Wilkinson, Madhura Bhave. People from the Spring community are also actively contributing to improving the project.
Suppliers	Spring Boot is built on top of Java SE and Spring Core.
Support staff	Spring Boot team is monitoring stackoverflow.com for questions tagged with spring-boot. Chat with the community happens on Gitter. Reporting bugs happens at their GitHub repository.
System administrators	Administrators of Spring Boot are the software engineers that have incorporated Spring Boot in their projects and products.
Testers	Large overlap with developers and maintainers. Andy Wilkinson and Phil Webb seem to be mostly contributing to the testing parts.
Users	A large user base in the Java world. Both individuals and large companies. Examples of companies: KLM Royal Dutch Airlines, ING, Philips.

Table 1 - Stakeholders of Spring Boot

24.5   Other stakeholders

- **Spring ecosystem stakeholders** - As Spring Boot is part of the whole ecosystem of Spring, other Spring projects (Spring Core, Spring Cloud, etc.) are important stakeholders that can affect the decision-making process for the framework.

- **Contributors** - Spring Boot is an Open Source system and various developers are contributing to the software.

- **System Architects** - Phil Webb is the lead architect of the Spring Boot project. Along with his colleagues at Pivotal, they are the ones forming the architecture of the framework.

- **Courses and instructors** - there are plenty of tutorials, courses and instructors that are helping the Java community integrate the project within their own applications. Some examples are Ken Kousen, Udemy, Baeldung and SpringFramework Guru.

- **Open source projects** - Spring Boot is extremely popular in the Java world. There are plenty of open source projects based on the framework. As such, they are interested in and dependent on the development of Spring Boot. Some examples are JHipster, Java Blogs Aggregator, Project Sagan. Another example can be found here.

24.6 Integrators and Decision-making process

Spring Boot is an avid example of an open-source system. It is backed up by a company (Pivotal) and is accepting contributions from anyone who complies with their code of conduct and signs a release form. All of the work is done in GitHub, and the mergers and maintainers of the code are members of the Pivotal team. The process analysis is based on some of the most commented issues and pull requests in the Spring Boot repository, described in Appendix A.

Most of the time contribution and decision-making follow a similar path. The work pipeline first requires an issue or a pull request (PR) creation from a team member or a contributor from the community. Contributors are strongly advised to create a small proof-of-concept project and create a pull request with the example in the spring-boot-issues repository. After that, the issue is either declined or approved.

The decision of what issues to be included, as well as which issues are relevant and should be pursued in a release seem somehow centralized. There is no evidence of a wide discussion for the general direction of the framework which seems to be decided internally at Pivotal. Issues added by contributors are mainly about bugs and smaller enhancements. The decision is also influenced by the number of people requesting a certain feature.

After approval, the PRs regarding the issue are merged after a discussion between the contributor and team members. Team members often lend a hand with polish and code standard compliance. Some issues can interfere or intersect with the work on other Spring projects (e.g. Spring Data, or Spring Cloud). In that case, representatives of those projects voice their concerns or comments. Issues might be declined if they are postponed for too long and become irrelevant, or if the Spring Boot team decides it is not in the direction they want to bring the product.

Work is done on release branches (e.g. 2.1.3.RC, 2.1.4.M2, etc.) following their release train. Those branches are regularly merged into the Master branch of the project. PRs are merged into the release branch by members of the team. Most often these members are Andy Wilkinson or Phil Webb, as well as Stéphane Nicoll. Phil Webb and Dave Syer are the project's creators and sometimes partake in the discussions.

24.7 Power-interest grid

After identifying the stakeholders for the Spring Boot project, a power-interest matrix was created. The grid classifies the stakeholders according to their power over the project and their interest in it.

Figure 1 - Spring Boot Power-Interest Diagram
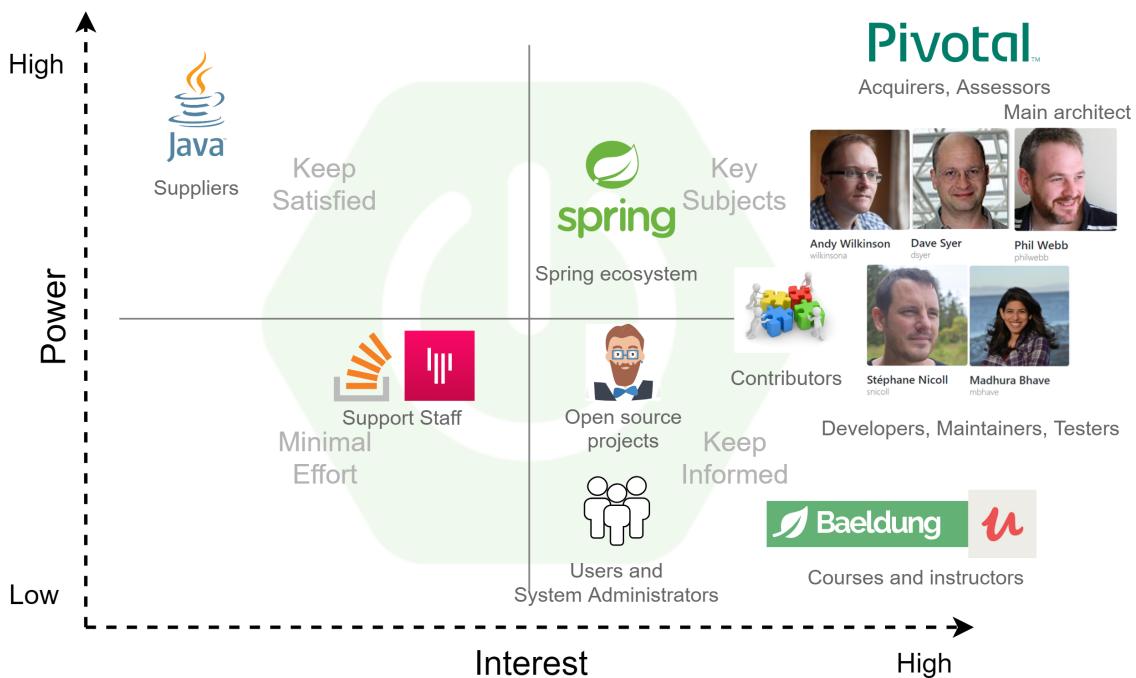
Figure 24.1: Spring Boot Power-Interest Grid

24.8 People to contact

After going through various sources of information regarding Spring Boot, our team came to the conclusion that the following three people would be the most interesting to contact:

Name	Reason to contact
Andy Wilkinson	Big role in the decision-making process
Phil Webb	Project lead and active on GitHub
Stéphane Nicoll	One of the most active software engineers in the project

Table 2 - People to contact regarding Spring Boot

24.9 Context view

In this section, the context view of Spring Boot will be discussed. According to Rozanski and Wood, "Context View describes the relationships, dependencies, and interactions between the system and its environment (the people, systems, and external entities with which it interacts)". It defines what are the system's capabilities and constraints and should be understandable to all stakeholders.

24.9.1 System Scope & Responsibilities

Spring Boot is a framework running on the Java Virtual Machine (JVM) and meant to ease the configuration process of Spring-based applications. It also addresses the learning curve of using the Spring ecosystem, which prohibited new developers from adopting it (Spring Framework #14521). This means that Spring Boot has excellent support for a vast range of Spring ecosystem projects such as Spring Data, Spring Security, Spring Cloud etc. Spring Boot includes the following capabilities, also described in the documentation and this article:

- Provide an opinionated and out-of-the-box Spring application with pre-selected configurations, but get out of the way once requirements start to diverge from the defaults.

All this makes Spring Boot a good starting point to develop production-ready Spring applications with minimal effort while making it easy to create custom modifications.

24.9.2 Context View Diagram

Context View Diagram can be seen in *Figure 2* with the entities further detailed below.
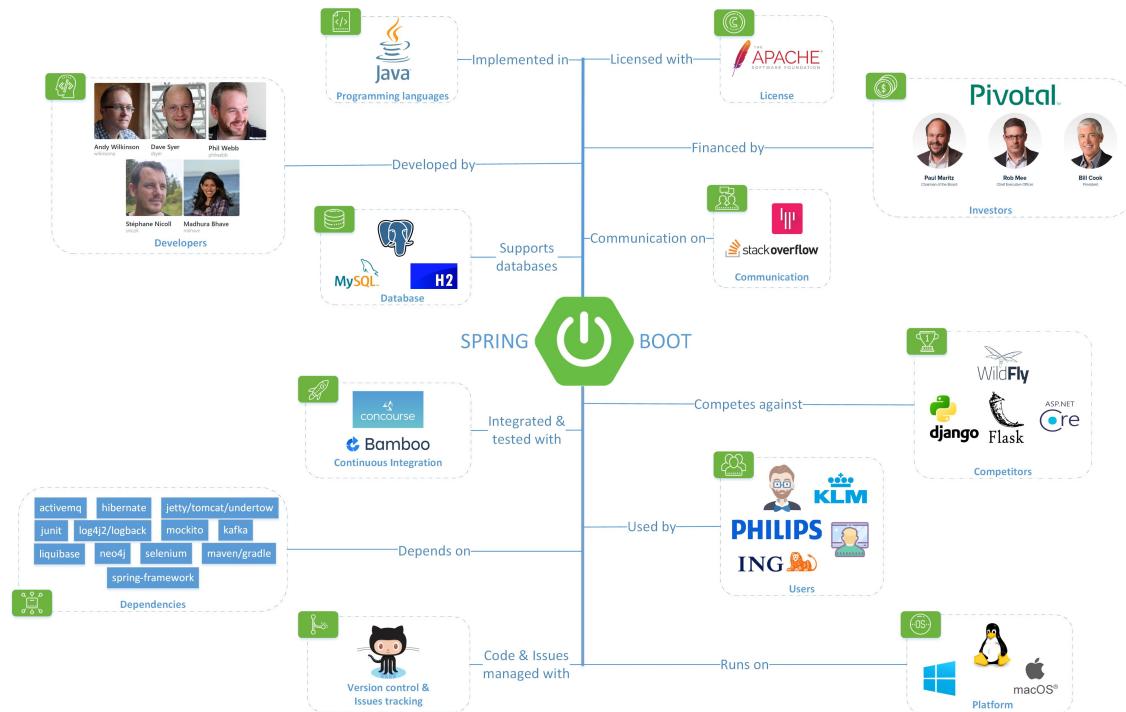
Figure 24.2: Context View

Figure 2 - Spring Boot Context Diagram

- **Languages** - Spring Boot is developed almost exclusively in Java (being an extension of the Spring Core Framework). According to the information found on Spring Boot GitHub Repository, the rest (HTML, Groovy, Javascript, Shell, Kotlin) make up less than 2% of the codebase.

- **Contributors** - Main contributors to this project are employees of Pivotal (most notably Andy Wilkinson, Stéphane Nicoll, Phil Webb, Dave Syer, and Madhura Bhave). Pull requests from third-party developers are accepted and encouraged.

- **Database Support** - Spring Boot supports embedded databases such as H2 but also accepts connection to a wide variety of databases through easily configurable JDBC drivers. Spring Boot also supports Spring Data which offers powerful object-mapping abstractions.

- **Investors** - Spring Boot is financed by Pivotal.

- **License** - Spring Boot is an open-source project released under the Apache 2.0 license.

- **Communication** - Open and informal discussion about Spring Boot is on Gitter. Spring Boot used to have a forum managed by Pivotal but they decided to move it over to Stackoverflow. Some of the communication is also happening on the GitHub issue tracker.

- **Issue Tracking** - Spring Boot's issues are tracked in GitHub issue tracker. People can also submit pull requests on GitHub.

- **Continuous Integration** - For CI, Spring Boot project uses Concourse CI and Bamboo.

- **Dependencies** - Even if the end-user will probably not be using all the dependencies, Boot project has to maintain all the options and therefore is dependent on a number of third-party software. Key dependencies include Maven/Gradle for dependency management, Tomcat/Jetty/Undertow for Java servlet containers and the Spring framework itself.

- **Platform** - Spring Boot is usually run on Windows, Ubuntu and other Linux based operating systems but it can run everywhere where Java Runtime Environment is available.

- **Users** - Spring Boot is used by individual developers to create minimum viable products (MVPs) as well as companies for Spring-based Web applications and microservices. This means that the users' skill level varies greatly. Spring Boot is also used as a core dependency by third-party open source software, such as JHipster.

- **Competitors** - For deploying Java microservices, WildFly is a competitor to Spring Boot, but it is not as widely used and has a smaller community. Django is a popular alternative in Python, which is more lightweight but may not have all the enterprise solutions Spring Boot offers. Other competitors include Flask (Python) and ASP.NET Core (C#).

24.10   Functional View

In the System Scope & Responsibilities section, we discussed what features and capabilities Spring Boot has. In this part, an overview is given of how the system interacts with its users and external entities. We try to present the Spring Boot in a way that people working with Java and the Spring ecosystem can get a general idea of what it is about.

Figure 3 - Functional Structure Model of Spring Boot

Spring Boot is comprised of multiple components. Their internal hierarchical structure and dependencies are discussed in detail in the Module Structure section of this chapter. The interactions between the different functional elements are shown in *Figure 3*.

Generally, the features provided by the project are aimed at the software developers who want to use the Spring ecosystem in their projects. It is a Rapid Application Development framework which allows for swift setup of a new project while allowing for customizations.

Functional Capabilities

A short walk through the initial parts of an application's lifecycle will explain where Spring Boot is positioned in the development process and how the project helps it. For the purpose of this example we assume that the developers have already chosen Java as their primary language and Spring Boot as the tool to kick-off the project.
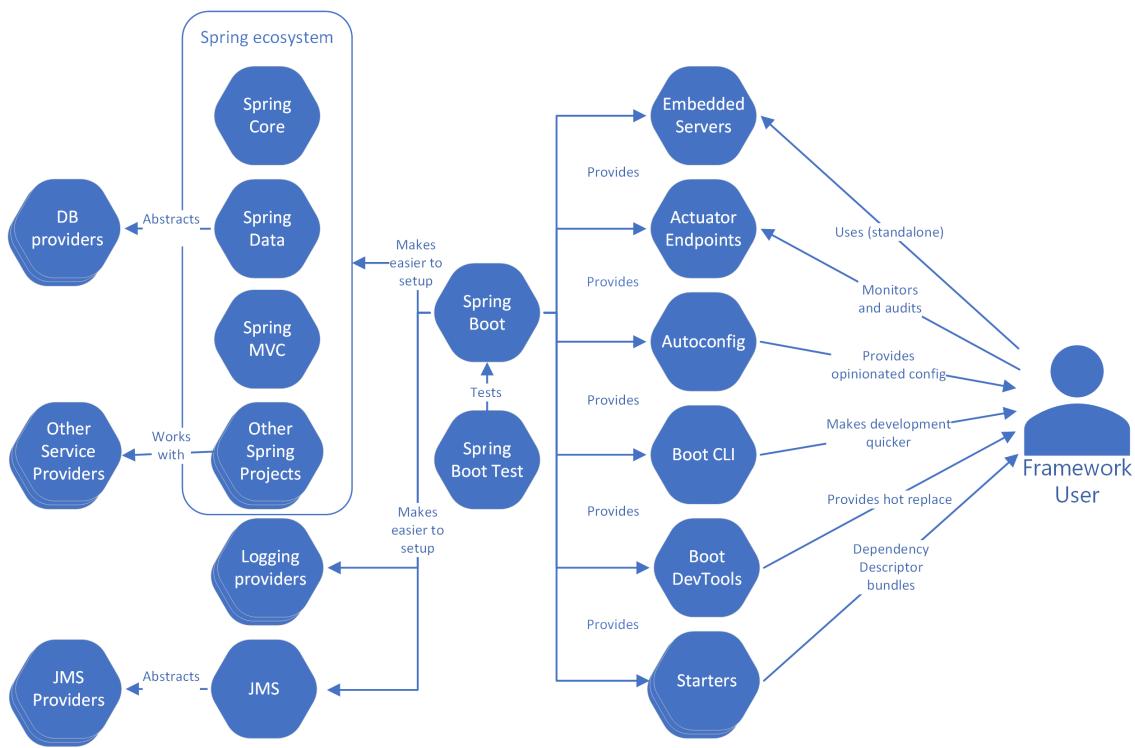
Figure 24.3: Functional Structure Model

For the sake of simplicity, the example project consists of a Database connection and a REST service.

The project is started by choosing one of the provided starters. They are conveniently bundled dependency descriptors used to simplify the dependency files. Developers can now include a single dependency for a certain functionality (e.g. working with an ORM) instead of the large number of dependency declarations needed otherwise.

The Spring Boot CLI can optionally be used to write Groovy scripts for fast prototyping and creating the project. It gives an alternative to just picking starters and writing boilerplate code in Java.

Developers can then opt-in to the auto-configuration functionality with a simple annotation. It is one of the most powerful time-saving tools in the Spring Boot toolset. Based on the dependencies on the classpath, the framework attempts to configure the application. For example, if `HSQLDB` is on the classpath, an in-memory database will be configured with default settings and values which are based on best practices and common usage models. Of course, Spring Boot always gives precedence to user-defined configurations over auto-configuration.

As the example project is a web-application which exposes a REST endpoint, a web-server is needed. Spring Boot provides embedded web-servers out of the box. The server is embedded in the resulting stand-alone `.jar`. As with other configurations, it is completely customizable.

In order to speed up the development process even further, quality-of-life improving tools module dev tools with features such as automatic restarts and resource caching can be used.

Using the provided tools, the example project can now expose a REST endpoint and store information in an in-memory database using only a few annotations and can be contained in a single executable `.jar` file. However, a running app needs to be monitored and audited. Spring Boot offers a solution to that as well with the Actuator. The Actuator provides endpoints such as `healthcheck` and `auditevents` to ease that process.

24.10.1   External Interfaces

Spring Boot makes it easier to start up and run a production-ready software quickly. However, most of the features (except the Actuator) it has, are provided by external sources. Creating a software application can be complicated but is helped by the various projects in the Spring ecosystem - from managing work with a database with the Spring Data projects to creating web applications with Spring MVC. All of those can be configured separately, but Spring Boot is very good at configuring and orchestrating them. It is created to alleviate the difficulties when creating an application and to allow developers to focus on the business problem they are facing.

As mentioned in the System Scope & Responsibilities Spring Boot is the answer to the critique of the Spring ecosystem and its steep learning curve. As such, its external interfaces are connected with the various Spring projects and indirectly with the service providers that some projects work with (e.g. different databases when using Spring Data).

24.11   Development View

24.11.1   Module Structure

Spring Boot is comprised of several modules, each serving a different purpose. A high-level overview of the module hierarchy is shown in *Figure 3*.
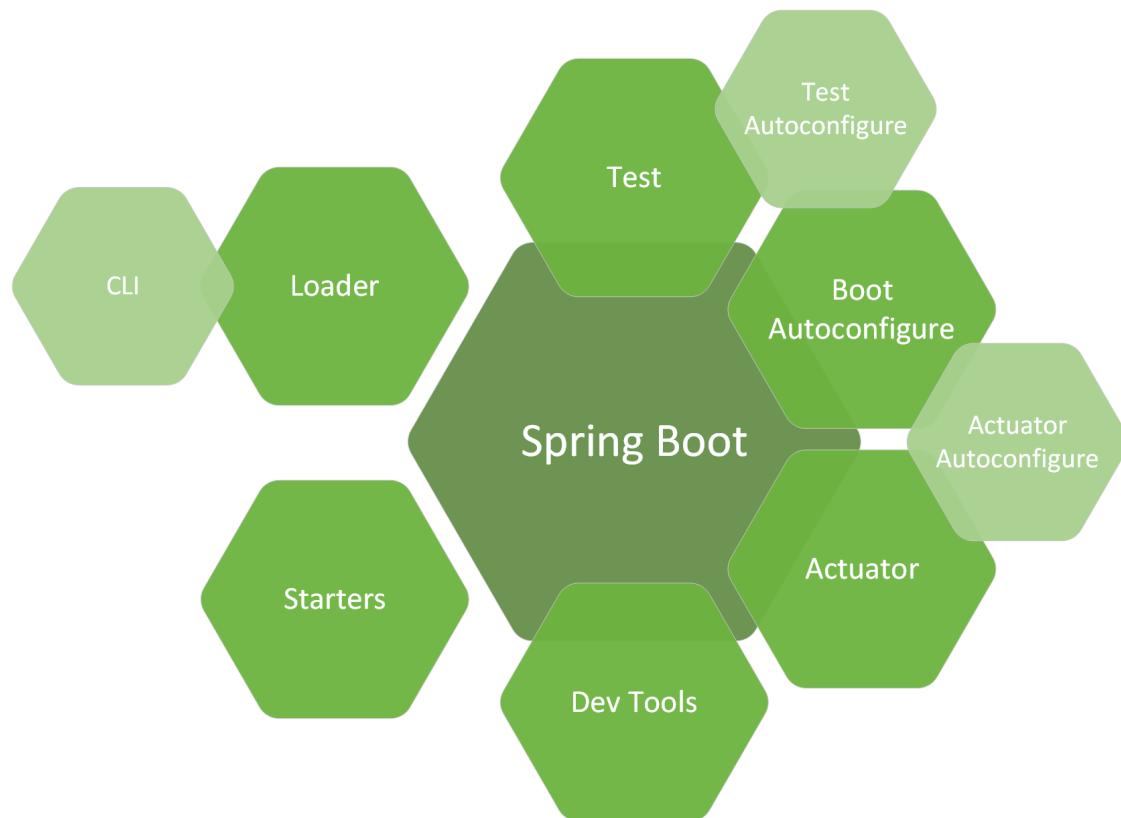
Figure 24.4: Module Structure

Figure 3 - Spring Boot high-level module structure. Overlapping indicates a dependency between modules.

- `spring-boot` is the main module, which provides features and methods that are used by the other modules of Spring Boot. For example, the ability to create applications with embedded web servers (e.g. Tomcat or Jetty).
- `spring-boot-autoconfigure` is responsible for the *opinionated* configuration of a Spring Boot application based on the contents of the classpath. It is also base for other `autoconfigure` endpoints.
- `spring-boot-actuator` provides the infrastructure and endpoints to monitor and audit applications.
- `spring-boot-actuator-autoconfigure` provides the configuration of different endpoints based on classpath content.
- `spring-boot-test` contains annotations and methods to write tests.

- `spring-boot-test-autoconfigure` automatically configures the tests and their dependencies based on what is present on the classpath.
- `spring-boot-devtools` is a toolset for quicker and smoother development with features such as automatic restarts.
- `spring-boot-loader` allows a Spring Boot application to be packaged in a single stand-alone Java archive (`.jar`) with all dependencies bundled (including web servers).
- `spring-boot-cli` is another developer oriented module providing a simple command line tool to quickly script and get an application up and running.
- `spring-boot-starters` are a set of modules providing prepackaged dependency descriptors so that developers need to manage only a single dependency.

24.11.2   Codeline Organization Model

For creating the Codeline Organization Model, we analyzed the project directory structure from the Spring Boot GitHub repository. The overall code structure of Spring Boot can be seen in *Figure 4*, with the main application folder `spring-boot-project` being further expanded.

Figure 4 - Spring Boot Code Structure

The **Functionality Part** of Spring Boot contains multiple modules that are grouped in the `spring-boot-project` folder. These modules were described in detail in the Module Structure. Each of the sub-folders (with a few exceptions such as `spring-boot-starters` which contains XML dependency descriptors) is organized into a `src` folder, which contains `main` and `test` folders for module source code and module unit tests (following the standard maven structure) and a separate `pom.xml` file among other properties and configuration files.

The **Development and Deployment Part** of Spring Boot is handled in `spring-boot-tests` for deployment (`/spring-boot-deployment-tests`) and integration (`/spring-boot-integration-tests`) tests. For Continuous Integration the Concourse pipeline scripts are grouped in `ci`. The folders `eclipse` and `idea` are used for providing support for Eclipse and IntelliJ IDEA to people that want to contribute.

The **Documentation Part** is contained in `spring-boot-samples` with Java sample applications and `spring-boot-cli/samples` with Groovy samples for the command line application. Javadoc and Asciidoctor documentation are found in `spring-boot-docs`. Other Asciidoctor files are placed in the root folder explaining the contribution pipeline, how to get support and so on. The repository root also contains the maven wrapper to ensure that the build has everything necessary to be run independently. By using the maven wrapper, `mvnw`, the reference documentation in HTML format can be built. The generated documentation can be found at `spring-boot-project/spring-boot-docs/target/generated-docs/reference/html`.

24.11.3   Common Design Model

In this section, part of the common design model is identified and discussed.

24.11.3.1   Standardization of Codeline

As previously mentioned in Codeline Organization Model, the actual program code and the unit test code are structured as follows:
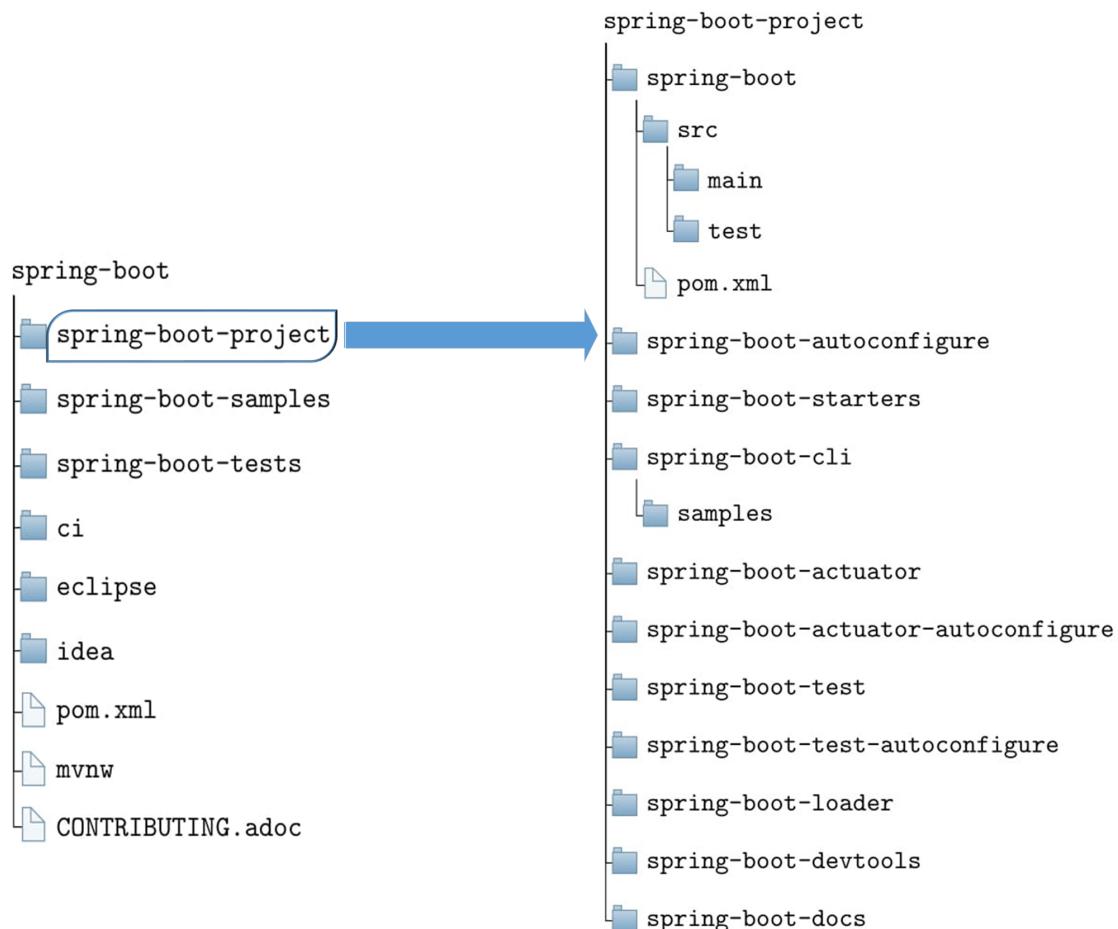
Figure 24.5: Code Structure

```
src/main/java/com/foo/Bar.java
src/test/java/com/foo/BarTests.java
```

Apart from the obvious codeline organization advantage, this ensures that unit tests are written separately from the mainline code, but still have access to package protected classes, methods, etc.

24.11.3.2 Object-Oriented Programming (OOP)

As most big Java projects, Spring Boot project uses the OOP paradigm. This has well-known advantages such as encapsulation, inheritance, and polymorphism. While conducting an interview with Stéphane Nicoll, he explicitly stated that the Spring Boot team tries to keep classes and classes contents as private as possible.

24.11.3.3 Dependency injection

Spring Boot uses the dependency injection technique. This allows enforcing the Inversion of Control principle which helps to increase the modularity of the project and make it more extensible. The Spring Boot project uses the `@Autowired` annotation in most cases for dependency injection which was introduced by the Spring Framework.

24.11.3.4 Modularity

When discussing with Stéphane Nicoll from the Spring Boot team, he stated that the Spring Boot project aims to have stand-alone modules which are not interdependent. The goal of this is to minimize coupling which increases maintainability of the code. While they make the code modular, they also wish to avoid making split packages.

24.11.3.5 Message logging

Spring Boot's logging relies on Commons Logging API for all internal logging and provides default configurations for Java Util Logging, Log4J2 and LogBack implementation of the API.

The default format for Spring Boot log messages is the following:

```
2014-03-05 10:57:51.112  INFO 45469 --- [           main] org.apache.catalina.core.StandardEngine  : St
2014-03-05 10:57:51.253  INFO 45469 --- [ost-startStop-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : In
```

By default, the logs are only displayed in the console. They can also be stored in a file, which requires configuration (e.g. in `application.properties`).

24.12   Testing and Release Models

24.12.1   Testing process

Integration tests, located in module spring-boot-tests (see Codeline Organization Model), are executed with every PR, as well as on every release. In addition, when a version is ready to be released, deployment tests (located in the same module) are also executed.

24.12.2   Release process

Spring Boot project is using Concourse for CI. The release versions of the project consist of milestones (*M#*), release candidates (*RC#*) and general availability releases (*RELEASE*) (expected to be stable and feature complete). There are also service releases, used for maintenance after a major release. Each release is available on repo.spring.io and GA are also available on Maven Central.

A prime example of the release process is the release of the major version 2.0.0 and can be seen in *Figure 5*.

```
2.0.0.BUILD-SNAPSHOT/    2017-11-17 12:20    -
2.0.0.M1/                2017-05-16 10:17    -
2.0.0.M2/                2017-06-16 10:40    -
2.0.0.M3/                2017-07-26 18:40    -
2.0.0.M4/                2017-09-15 07:20    -
2.0.0.M5/                2017-10-12 01:40    -
2.0.0.M6/                2017-11-06 07:00    -
2.0.0.M7/                2017-11-30 05:40    -
2.0.0.RC1/               2018-01-31 07:00    -
2.0.0.RC2/               2018-02-21 14:00    -
2.0.0.RELEASE/           2018-03-01 01:00    -
```

Figure 24.6: Spring Boot release process

Figure 5 - Spring Boot release process for the major version 2.0.0

Like all the other projects in the Spring ecosystem, there is a strong set of rules for versioning the releases of Spring Boot. This is not just following best practices, but the community download page depends on the naming and semantic details of the release artifacts.

Scripts for releasing a version can be found in the ci module. Each version is accompanied by release notes which can be found in the Wiki section of the GitHub repository. Release notes contain information on how to upgrade from a previous version, new noteworthy features, as well as various deprecations.

24.13 Technical debt

[Technical debt](#) is a concept in programming that reflects the extra development work that arises when code that is easy to implement in the short run is used instead of applying the best overall solution.

24.13.1 Identifying technical debt

Identifying technical debt in a project is an extremely challenging task. We used automated code inspection tools, but they can only support the identification process. Our team interviewed one of the leading software engineers of Spring Boot - Stéphane Nicoll - who gave us valuable insights on how big the technical debt is and what does the project's team do to keep it to a minimum.

24.13.2 Static code analysis

Static code analysis is done using SonarQube, to understand the technical debt of Spring Boot.

There are four main elements that the static code analysis tool emphasizes on: - **Bugs and vulnerabilities** - Most of the bugs identified by SonarQube are minor. Still, the tool gives Spring Boot a Reliability Rating `E`, which stands for at least one blocker bug in the code, shown in *Figure 6*.

```
188                 private KeyStore loadStore(String type, String provider, String resource,
189  ⌃                            String password) throws Exception {
```

Define and throw a dedicated exception instead of using a generic one. ▪▪▪   2 months ago ▾ L189 🔗
⊗ Code Smell ⌃ Major ○ Open Not assigned 20min effort   🏷 cert, cwe, error-handling

```
190                     type = (type != null) ? type : "JKS";
191  …                   KeyStore store = (provider != null) ? KeyStore.getInstance(type, provider)
192                                    : KeyStore.getInstance(type);
193  …                   try {
194  …                         URL url = ResourceUtils.getURL(resource);
195  …  ❗                      store.load(url.openStream(),
```

Use try-with-resources or close this "InputStream" in a "finally" clause. ▪▪▪   2 months ago ▾ L195 🔗
🐛 Bug ❗ Blocker ○ Open Not assigned 5min effort   🏷 cert, cwe, denial-of-service, leak

```
196                                    (password != null) ? password.toCharArray() : null);
197  …                   return store;
198                   }
199  …               catch (Exception ex) {
200                         throw new WebServerException("Could not load key store '" + resource + "'",
201                               ex);
202  …               }
203           }
```

Figure 24.7: Critical issue according to SonarQube

Figure 6 - Critical issue - Not closing an input stream can lead to memory leaks.

- **Code smells** - Technical debt ratio of the Spring Boot project is less than `5.0%`, according to SonarQube. This result indicates that the Spring Boot team takes quality in a serious manner as was confirmed by the conducted interview.

- **Duplications** - According to SonarQube the code duplication is only `0.7%` hence it is not a big problem for the framework.

- **Cyclomatic complexity** - Using SonarLint and manual code inspection, we have identified several cases of high Cyclomatic complexity [4]. An example can be seen in *Figure 7* where the recommended value is exceeded.

```
@Override
public Object getProperty(String name) {
    if (StringUtils.hasLength(name)) {
        for (MappedEnum<?> mappedEnum : MAPPED_ENUMS) {
            if (name.startsWith(mappedEnum.getPrefix())) {
                String enumName = name.substring(mappedEnum.getPrefix().length());
                for (Enum<?> ansiEnum : mappedEnum.getEnums()) {
                    if (ansiEnum.name().equals(enumName)) {
                        if (this.encode) {
                            return AnsiOutput.encode((AnsiElement) ansiEnum);
                        }
                        return ansiEnum;
                    }
                }
            }
        }
    }
    return null;
}
```

Figure 24.8: Cyclomatic complexity in Spring Boot

Figure 7 - High cyclomatic complexity (21) in Spring Boot

Spring Boot overall has a small amount of technical debt. Some of it is known to the development team. This deliberate debt is placed in a separate milestone. The issues in this backlog are not yet scheduled for a particular release and are mostly bugs or enhancements that Spring Boot team wants to solve in a future version.

24.13.3   Insights from Stéphane Nicoll

Spring Boot's core team engineer Stéphane Nicoll discussed the technical debt of the project in details in the interview we did. In this section, we put some of the most exciting findings according to our team:

- **Major refactoring** - Spring Boot team had to perform a significant refactoring of the `actuator module` once Spring Framework implemented the reactive paradigm. The module heavily relied on `Spring MVC` that uses `servlet containers`, which is not compatible with the new reactive style. The team decided to rewrite the whole module from scratch but kept its public API.

- **For team attention** - we have noticed flagging of issues with a label `for-team-attention`. Once an issue is created (either from a team member or from a contributor), a team member can label it. The team holds two meetings every week. During those meetings, the person who marked an issue

---

[4]McCabe, Thomas J. "A complexity measure." IEEE Transactions on software Engineering 4 (1976): 308-320.

explains the reasoning behind it, and then the participants in the meeting collectively decide how to handle the problem.

Stéphane Nicoll confirmed our finding that the Spring Boot team tries to keep the technical debt to a minimum and is building its code base with great care for readability and quality. A remarkable fact is that the Spring Boot team does not use a technical debt identification tool, apart from the static code analysis tool provided by IDEs. As Stéphane stated, "we don't use any tool, we don't see any benefit from using one". The team works "in an organic way", relies mostly on experience, strong sense of belonging to the project and is heavily influenced by the Spring community. In his words:

> If we do something wrong, the community will kick our asses anyway.

24.14   Testing Debt

Issue analysis (#13526 and #14684) prior to interview with Stéphane Nicoll made us believe that there are no code coverage reports in CI tools. This was later confirmed while conducting the interview.

The project has an advanced testing environment with embedded Docker containers (#10516) and Servlets which makes test coverage of the project high. The test suite is simple to run on the command line with `./mvnw verify`.

IntelliJ IDEA built-in code coverage tool was used to run unit tests on most important modules, which were `spring-boot`, `spring-boot-actuator`, `spring-boot-actuator-autoconfigure` and `spring-boot-autoconfigure`. The results show that those modules are very well tested with class and line coverage being mostly over 90% (see *Table 4*).

Module	Class Coverage	Line Coverage
spring-boot	93%	90%
spring-boot-actuator	94%	93%
spring-boot-actuator-autoconfigure	95%	90%
spring-boot-autoconfigure	89%	93%

Table 4 - Test coverage for Spring Boot's main modules

Therefore it can be said that testing debt of Spring Boot is minimal and will not affect the project negatively if kept at this level. However, the project could make use of automated tooling to report on potential testing debt and make this information available to contributors.

24.15   Evolution of technical debt

In order to "calculate the trends of the technical debt time series" [5] for the Spring Boot project, we first need to take a look at the history of the whole Spring Framework. The evolution of Spring Framework from interface21 to SpringSource and finally Pivotal is displayed in *Figure 8*.

---

[5]Digkas, Georgios, et al. "The evolution of technical debt in the apache ecosystem." European Conference on Software Architecture. Springer, Cham, 2017.
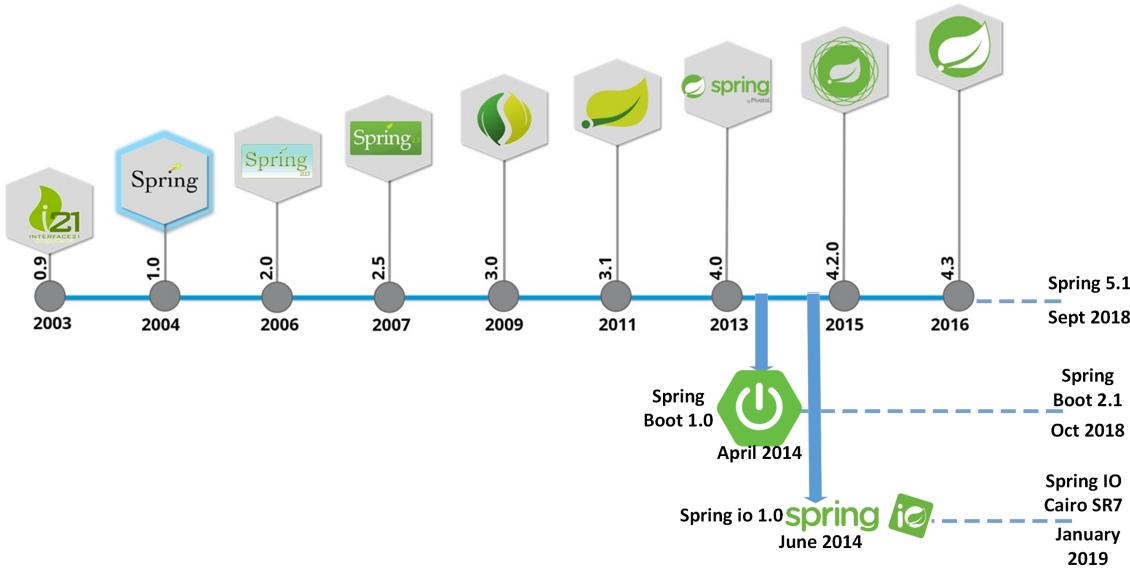
Figure 24.9: Spring History

Figure 8 - History of Spring Framework (adapted from here)

Spring Boot was launched in April 2014 due to the need for support for containerless web application architectures in Spring Framework with Phil Webb as lead architect, after Pivotal was created in 2013. Due to this, Spring Boot was not affected by a major change in management that could determine Technical Debt arising from different performance management styles.

Another aspect related to the evolution of technical debt from the Spring world is the launch of Spring IO in June 2014. Generally, as a release pipeline, a new Spring Framework version will trigger a new Spring Boot release followed by a Spring IO platform release which incorporates the new version of Spring Boot. The Technical Debt arising from here is related to the decreasing difference between Spring Boot's and the platform's dependency management. That is the reason why Spring IO Platform end-of-life was announced with Cairo being the last maintained version.

The fact that "software process evolution has a clear effect on technical debt" [6] was proved in Spring Boot project by a major switch. When Spring Boot decided to replace their CI platform from Bamboo to Concourse, they benefited from having a platform maintained by Pivotal. The migration to Concourse was problematic, but it was a known fact to the developers and they took the effort to remove the debt later on.

From its conception until today Spring Boot has had steady growth in the code base (as seen in *Figure 9*) but we did not manage to find any major technical debt issues.

Figure 9 - Evolution of code size (Made using git-of-theseus)

---

[6]Yli-Huumo, Jesse, Andrey Maglyas, and Kari Smolander. "The Effects of Software Process Evolution to Technical Debt—Perceptions from Three Large Software Projects." Managing Software Process Evolution. Springer, Cham, 2016. 305-327.
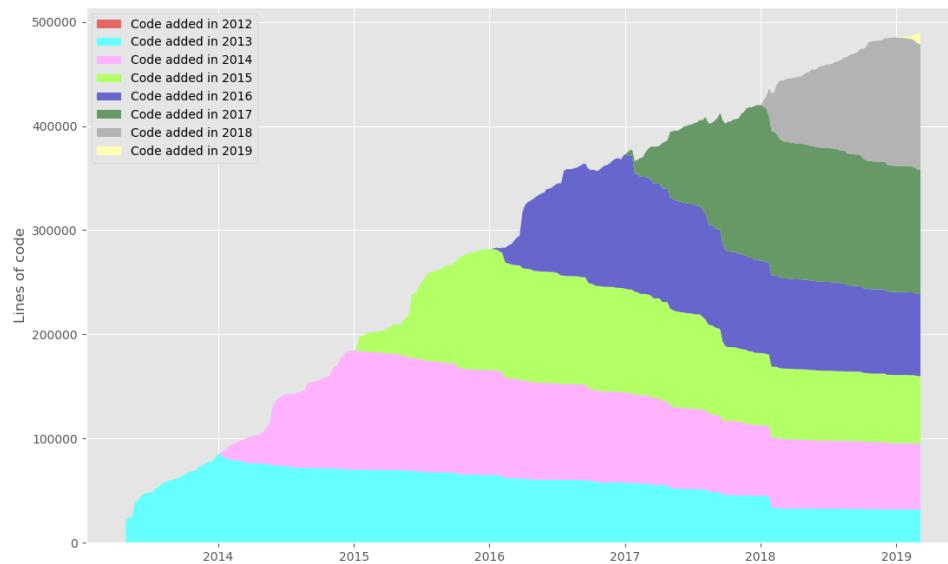
Figure 24.10: Code Size Evolution

24.16 Discussions about technical debt

Spring Boot's team generally uses Labels to identify and characterize issues. There are currently 29 labels, but none of them is directly related to technical debt (e.g. `technical-debt`, or `refactoring`).

On the other hand, when searching through the issues on GitHub, there are less than 90 issues out of more than 16000 issues in total mentioning technical debt or refactoring. Usually, these are proposing some behavioral changes (for example #14680). There are some issues created by the team with known technical debt (#14412). However, the maintainers are very cautious to make major changes, as they might be breaking ones to some end-users that have already adopted the existing functionality.

`TODO`s and `FIXME`s for the whole project are an almost negligible amount. They mostly relate to testing scenarios rather than actual functionality. There are only two examples of `FIXME` that are in the mainline code that are not about changing the name of a method.

The cleanliness of the code is helped by the reviews of team members before merging. Another reason is the provided code style guidelines and schemes for IntelliJ IDEA and Eclipse, which disallow comments in the code, as well as other polluting elements like unused imports.

24.17 Possible Improvements

We chose to propose two improvements to the Externalized Configuration functionality, which now allows expressions instead of a fixed list of profiles defined by the users.

Stéphane Nicoll pointed out that after introducing expressions for profiles, the changes have caused quite a few bugs and they have not decided on how to tackle them. Analyzing the GitHub issues confirmed the technical debt.

The first problem is the wrong order of profile overriding. We suggest deciding on an intuitive solution on profile configuration ordering that is based on industry standards. Specifically, old GitHub issue #3845 indicates that instance specific values (outside JAR) should override environment (i.e. PROD, TEST, etc.) specific properties embedded in the application (JAR). Hence, in current configuration ordering, this:

```
1. Command line arguments.
...
7. Profile-specific application properties packaged inside your jar (application-{profile}.properties a
8. Application properties outside of your packaged jar (application.properties and YAML variants).
9...
```

should become this:

```
1. Command line arguments.
...
7. Application properties outside of your packaged jar (application.properties and YAML variants).
8. Profile-specific application properties packaged inside your jar (application-{profile}.properties a
9...
```

where configuration with a smaller number will override the one with a higher number.

The second problem is related to referencing profiles. In Spring Boot configuration, you can use `spring.profiles.include` keyword to add different application profiles (other configurations). The problem arises when profiles with higher ordering than the original configuration file, are referenced. In this specific case, the referenced profile has no effect, and this behavior is unexpected by the users.

The issue can be solved by giving `spring.profiles.include` special privileges which would combine values across all configuration files. This solution was also suggested in #16023.

The proposed changes would be breaking ones, but they are eventually unavoidable and deal with the technical debt to avoid future complications regarding profiles in Spring Boot.

Our team has also actively contributed to the Spring Boot project as shown in Appendix B.

24.18   Conclusions

This chapter provides an overview of the Spring Boot project whose purpose is to simplify the work with Spring Framework. We analyzed Spring Boot's Stakeholders, Context View, Functional View, Development View and Technical Debt.

During our work, we found that the Spring Boot team is really community oriented which was confirmed by a video call with Spring Boot core developer, Stéphane Nicoll.

The project's architecture is divided into independent modules. While they do not have a list of requirements to follow specific coding principles, such as SOLID, they rely on experience and pragmatic thinking to apply good coding practices along with strict contribution guidelines.

Looking into Technical Debt, we found that Spring Boot has good test coverage. While analyzing issues, the SonarQube report and interviewing Stéphane Nicoll, we concluded that the project also has a very small technical debt. To our surprise, they use no tools for code coverage other than the ones provided by the IDEs.

Overall, it can be said that Spring Boot is heavily used by the Java community and the design decisions are taken with great care. Therefore Spring Boot will undoubtedly remain as one of the go-to choices for creating web and enterprise applications with Java.

24.19   Bibliography

24.20   Appendix A

A compilation of researched issues and pull requests from the Spring Boot repository is presented below.

PR	Context	Final Decision

`ASCII art banner` generated from an image	Big discussion about adding ASCII image to the loading screen or not, with a rather humorous tone. No real discussion of whether or not it should be merged - all opinions are positive. Proposed by a third-party committer.	Unanimously accepted by all parties, i.e. team members, contributors and members of other Spring teams. Merged by Phil Webb for release 1.4.0.M2
Add `Quartz Scheduler` support	The issue was proposed by an active contributor (vpavic). It was met with a big positive response from the community (many +1s) and the team members.	It took more than 1.5 years for it to be approved and merged. Merged in version v2.1.3.RELEASE by Andy Wilkinson in 2017.
Make `run_user` for the launch script configurable	The creator is a first-timer and he created a PR and an issue. The team marked the issue as a duplicate to the PR with the comment "no need to create an issue and a PR. A PR is enough, thanks!". Discussion about the PR is straightforward and only minor comments are made by the team and some contributors. Note: If someone is too slow to respond, the ticket is passed down to other people	It took exactly 1 year to be merged in release 2.1.0.M3. Merger: Andy Wilkinson
Add auto-configuration for `WebServiceTemplate`	The issuer (first-timer) created a PR directly without an issue. There was no answer from the team, so he persisted and pushed them for one. After that, a back-and-forth communication between the contributor and the team starts in order to fix code issues. Part of the discussion is whether or not this feature should be part of Spring Boot, or another Spring project.	In the end, the PR is merged after only 2 months in v2.1.0.M1 Merger: Stéphane Nicoll
Introduce `HealthIndicatorRegistry`	An issue is created. A discussion ensues what might be some use cases for the proposal. After that, this PR is created. There is a discussion about the code.	The merging is postponed for after the major change to add reactive streams to the code base of the project. After that, the issue is merged. Merger: Stéphane Nicoll Time needed: 2.5 years

Provide a `@DataMongoTest` similar to `@DataJpaTest`	A simple technical discussion. Note: Apparently PR is interchangeably used with an issue in GitHub. The interesting thing here is that team members merge suggestions from people, however, they might add some polishing to the PR (with consent from the author of the PR, of course).	Merged after only a couple of months in January 2017 Merger: Andy Wilkinson Release: v1.5.0.RELEASE
Add `Spring Data Neo4j` support	Technical discussion. The Spring team member was not very familiar with the proposed technology, so there was a bit of an explanation phase going on. Stakeholders from other projects pitched in (neo4j and Spring Data) with some knowledge.	After discussing with the team members of Spring Data and Neo4J, enough context was gained by the Spring Boot team to accept this change. Merged by Stéphane Nicoll Release: v1.4.0.M2
Add `Couchbase` support	There is a technical discussion between the Spring boot team and the contributor. Members of other Spring projects (Spring Data) pointed out that the target component is being rewritten and this change should wait. "Since the `spring-data-couchbase` module is in the process of being rewritten and ported to Couchbase SDK 2.x, I think it'd be better to wait for it before starting integration in Spring Boot. See (work in progress)" - Simon Basle	After waiting for the next major release, the PR is updated with the new changes based on changes of adjacent components. All stakeholders are happy and the PR is merged by Stéphane Nicoll Release: 1.4.0.M1
Added `Cassandra` support for Spring Boot	Just large technical discussion between the team members and the contributor. A lot of comments from the team members about the code. A contributor asks for status. Otherwise, nothing particularly interesting.	Merged by Phil Webb Release: v1.3.0.RC1

Add support for configuring 'lobgack' logging via JMX	A contributor wants to make an enhancement in the logging module. Wants to make it in the 1.3.1.RELEASE. PR is opened on Nov 14, 2015. The PR is closed in favour of more general logging endpoint. Spring Boot team realizes there is a similar feature in Spring Cloud which they wish to integrate for version 2.0. PR is reopened because the integration with Spring Cloud would happen in release 2.0. After a long and large technical discussion, the contributor and Spring Boot team realize they need a public API from logback to finish the ticket. Since there is no movement from logback side - the PR is closed.	PR has been opened, then closed and then reopened. First, Spring Boot team did not want to make the configuration too specific (for logback only) - they wanted a generic endpoint. After a new PR was opened by the contributor for the generic endpoint, Spring Boot team realizes such a thing exists in Spring Cloud and would be integrated into a later version. PR was reopened but the team realized they need a public API provided by logback. A ticket was opened in logback Jira. Stéphane Nicoll closed the PR on May 18, 2018, as there was no progress with the logback ticket.
Fix issue #1668: Added support for `custom system class loader` to enable loading of various classes that are packaged inside nested jars at the system level	A contributor opens this PR on Oct 7, 2014. Other contributors seem also interested in the progress of the PR. A large discussion is raised of how to test the enhancement. There seems to be no progress on it.	Closed by Andy Wilkison on Jan 8, 2016, after internal discussion. Spring Boot team has decided this is an edge case and not needed by the vast majority of Spring Boot users.
Add `Neo4J` auto-configuration	PR created by a contributor on Sep 4, 2015. The main topic in the comments - integration test the sample. Due to other pressing issues, this PR is postponed for the next release. With the new Spring Boot release, there is an update that needs to be integrated into the current PR. The contributor creates a new PR instead of updating the current one. Current one becomes obsolete.	Stéphane Nicoll closed the PR on Mar 22, 2016. Marked as duplicated in favour of Spring Data Neo4j support. Reason - it duplicated a newly create PR.

Delegate `command line` `system properties to` `spring-boot-gradle-plugin` `runApp` task	PR opened on Mar 27, 2014, by a contributor. Apparently, the issue was introduced with a specific version of Spring Boot. Technical discussion about how to do this. A few other contributors jump in to help with code. Phil Webb closed the PR after the statement that it should be done the way Grails do it. After the PR is closed, there is still a discussion on whether the improvement is needed or not. Andy Wilkinson clarifies at the end that in the 2.0 snapshot, the configuration is back to being the same as in 1.x.	Closed by Phil Webb on Jan 11, 2017. Reason - Spring Boot team prefers to follow the Grails convention of passing properties, even though the proposed one is easier.
Make sure that starting a `JMS` connection does not block infinitely	PR opened by a contributor on Oct 31, 2017. The change is too large for entering the current release, also due to the fact that Spring Boot team needs to support Java 6. Later on, the contributor dives further into the problem. Together with the Spring Boot team, they discover that the approach to fixing the issue should be different. A new PR was created and the current one was closed.	Closed by Andy Wilkinson on Nov 15, 2017. Reason - the approach to fixing this issue should be different. A new PR is created for that purpose.
Add `MessagePack` support	Created by a contributor on Nov 21, 2015. On Jan 6, 2016, the PR was marked with status "waiting-for-votes" to see what is the demand for this ticket. One of the comments expresses concern that it is weird to wait for votes to merge the issue. Response from Andy Wilkinson is that ongoing cost needs to be considered after it being merged (more code to maintain, another dependency, another feature for users to understand). In the end, PR is closed because Jackson 2.9 is not supported yet.	Closed by Toshiaki Maki on Oct 13, 2018. Reason - MessagePack hasn't supported Jackson 2.9 yet.

Customize `compression` only when it is enabled	Created by a contributor on Oct 12, 2018. A discussion arises whether the issue exists in the current version (2.x) or only in an old one (1.5.x).	Phil Webb closed this PR on Oct 15, 2018. It is an issue only in 1.5.x and Spring Boot team is trying to keep changes there to a minimum.
Add `@DataElasticSearchTest` support	Introduction of a new annotation by a contributor on Mar 15, 2017. The changes were initially approved. After technical discussion, the PR was polished. Andy Wilkinson jumps into the discussion and points out that ElasticSearch have dropped support of a feature needed for this ticket. He also states that due to this, there are severe restrictions regarding integration tests.	Andy Wilkinson closed the issue on Feb 2, 2018. Statement - it seems that the situation with the limitation due to ElasticSearch tests won't change in the foreseeable future.
Expose `Hibernate` Statistics	Created on Jul 24, 2015 to fix issue #2157. Spring Boot team could not figure out a way how to integrate integration tests for Spring Boot 5, which delayed this issues. Two year later, the statistics were move to micrometer and this PR was closed.	Stéphane Nicoll closed this PR on Oct 3, 2017. Reason - statistics were moved to micrometer team. The contributor should discuss the changes with that team now.
Add option to configure `logging filters`	Enhancement PR, created on Jul 9, 2016. The ticket was moved back and forward for improvements. Andy had some concerns it was not the best approach to tackle the problem. The contributor said he would implement it but could not do it within reasonable time and the PR was closed.	Stéphane Nicoll closed this PR on Feb 20, 2019. Reason - issue was moved back and forward without any progress, it could be reconsidered again in the future.

24.21   Appendix B

Our team's contribution to the Spring Boot project is presented in the table below.

PR	Status	Release version

#16327: Add Spring-specific styling to Gradle Plugin's documentation	Merged	2.2.0.M2
#16326: Add Spring-specific styling to Actuator's API documentation	Merged	2.2.0.M2
#16174: Replace ${sys:PID} with %pid in log4j2 configuration	Merged	2.2.0.M2
#16166: Update ambiguous documentation about ConfigurationProperties	Merged	2.1.4
#16182: Determine Spring Boot version on startup correctly when using Jigsaw	Merged	2.2.0.M3

Our team is represented in the Spring Boot 2.1.4 release with Update ambiguous documentation about ConfigurationProperties #16166.

Chapter 25

Terraform


```
resource "student" martijn.de.heus" {        resource "student""mayke.kloppenburg"
     student_number   = "4367839"                  student_number    = "4383265"
     github_id          = "martijndeheus"          github_id          = "mayke93"
}                                             }

resource "student" "marlo.ploemen" {          resource "student" "jan.scheurer" {
     student_number   = "4276906"                  student_number    = "4301633"
     github_id          = "emptyless"              github_id          = "janscheurer"
}                                             }
```

Delft University of Technology

Terraform is a tool for writing infrastructure as code. It makes it possible to code infrastructure in HCL (Hashicorp Configuration Language) for many different cloud providers. The project itself is open-source

and written in Go. The purpose of this report is to provide an overview of the Terraform's architecture from different perspectives: stakeholders, context, developers, technical debt and usability.

Terraform was researched by looking for information online, analyzing GitHub pull requests, analyzing code manually, with tools, and doing an interview with a day-to-day user.

The stakeholders and their roles were identified, along with the integrators of Terraform. The context view shows Terraform with all services it facilitates or uses. In the development view the modules of Terraform are outlined and standardized practices are described. Technical debt and its history over different Terraform versions was identified. Lastly, this report looks at the usability of Terraform.

The Terraform project has many strong points and, in our opinion, also some flaws. A strong point is that the project is written with modules providing separation of responsibility. Flaws were mainly found for technical debt department, where good practice is not always adhered to. Another good point of Terraform is that it makes collaborating on infrastructure configurations an easy job.

25.1   Table of Contents

25.2   Abstract

Back in the days businesses built their IT infrastructure in large air conditioned server rooms. Nowadays, there is a much easier option; use resources from cloud providers such as Amazon, Microsoft or Google. It is no longer necessary to have the hardware in-house and configure them manually. We can now even write Infrastructure as Code (IaC). Terraform is an open source project that is used for this purpose. Instead of learning to code one or more cloud specific languages, Terraform presents the possibility to code infrastructure for many different cloud providers in HCL. Currently, the stable version is v0.11.0, but v0.12.0-beta1 is also out.

In this report, the architecture of Terraform will be discussed from various viewpoints. Firstly, the stakeholders and their interests will be identified and described. Then, information about Terraform's context will be given, partly via a model. The development view summarizes interesting details for developers who want to work with Terraform. Next, 'technical debt' will go into several different types of debt and the history of them. Lastly, Terraform will be discussed from a usability perspective. Some details will be left out of the report to allow the report to focus on the most important aspects. These appendices can be found in the appendices folder on GitLab.

25.3 Stakeholders

25.3.1 Stakeholder identification

Firstly, the stakeholders will be identified. Rozanski and Woods identified 11 classes of stakeholders [1]. The table below presents these classes and describes the corresponding stakeholders for the Terraform project. Three classes, which are less interesting, are left out and can be found in appendix A.

Class	Stakeholder	Description
Acquirers	HashiCorp	HashiCorp is the owner of Terraform. HashiCorp is a company that develops (open-source) software to provision, secure, connect, and run any infrastructure easily [2]. The board of HashiCorp is responsible for creating their ToA in which the foundations for their vision, roadmap and product design are presented. HashiCorp also attain ownership of all contributions made to the system through Github by requiring contributors to sign their CLA.
Assessors	HashiCorp and core team	Since HashiCorp as a corporation is the publisher of Terraform, the board is likely involved in ensuring legal and regulatory compliance. This has to be done in collaboration with the core team of Terraform, as they are responsible for day-to-day desicion-making regarding the development.

---

[1]Rozanski, Nick, and Eoin Woods. 2011. Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives. 2nd ed. Addison-Wesley Professional.

Class	Stakeholder	Description
Communicators	Core team and contributors	The core team of Terraform published the Terraform website, including the docs and guides, to their GitHub along with the source code of Terraform. This allows for contributors to help maintain their documentation and even add guides. The core team published guidelines with the aim to structure the communication on GitHub in order improve efficiency. Companies that offer services which integrate with Terraform also offer communication and training, as an example Microsoft has published a video series on how to use Terraform with Azure.
Developers	Core team and contributors	Terraform is developed by contributors and the core team in collaboration via their GitHub.
Maintainers	Core team	Because HashiCorp offers their products as a service to customers they have a huge stake in maintaining the source code. They will do so independent from open-source contributors, even though they are of course welcome to help.
Suppliers	AWS, GCP, Azure, etc.	Terraform is built on suppliers of resources. The main suppliers are the following: Microsoft Azure, Google Cloud Platform and Amazon Web Services. Besides these Terraform can be used with many different suppliers ranging from Infrastructure as a Service (e.g. AWS, GCP, Azure) to Platform as a Service (e.g. Heroku).
Testers	Core team and contributors	HashiCorp requires contributers to write their own tests when they add features with a new pull request, however because many contributions are made by the core team, they can be classified as testers themselves as well.

Class	Stakeholder	Description
Users	Variety of companies	The system is used by different companies, including: Barclays, ITV and SAP Ariba. Barclays even proudly writes they value acting as customer advisor and contributors to Terraform.

25.3.2 Additional stakeholder type identification

After looking into the different types of stakeholders identified by Rozanski and Woods, additional stakeholder types which are specific to the Terraform project are identified. These stakeholder types are presented and described in the following table:

Class	Stakeholder	Description
Cloud-specific competitors	AWS Cloud Formation, Azure Resource Manager, Google Cloud Deployment Manager, etc.	Many suppliers we identified in the previous table offer Infrastructure as Code tooling similar to Terraform specifically for their infrastructure products. This leads to interesting dynamics since these companies now are both suppliers and offer a competing product.
Cloud-agnostic competitors	Chef, Puppet, Saltstack, etc.	Even though these products work differently from Terraform (and can even be used together), we still list them since they offer companies different methods to create 'Infrastructure as Code'. The main difference is between Configuration Management (e.g. Terraform and CloudFormation) and Orchestration Management (e.g. Chef, Puppet and Saltstack).

---

[2]HashiCorp. "About Hashicorp." 2018. https://www.hashicorp.com/.

Class	Stakeholder	Description
Partners	List of companies	Different companies partner up with HashiCorp to 'expand their technical skills and go-to-market initiatives around DevOps principles, cloud technologies, and data centre management by leveraging the differentiated offerings of the HashiCorp product suite.' [3]

25.3.3   Stakeholder identification summary

To sum up the identification of the stakeholders involved in this project we present a table which shows the involved actors and the different roles they fulfil in the project.

Nr.	Stakeholder	Roles
1	HashiCorp	Acquires, assessors
2	Core team	Communicators, developers, maintainers, support staff, testers
3	Contributors	Communicators, developers, testers
4	AWS, GCP, Azure, etc.	Suppliers, cloud-specific competitors, partners
5	Chef, Puppet, Saltstack, etc.	Cloud-agnostic competitors
6	Customers	Users
7	HashiCorp customer service	Support staff
8	Sysadmin teams (HashiCorp and external)	System administrators
9	Production Engineering teams (HashiCorp and external)	Production Engineers

Stakeholders 1 through 6 will be analysed in the stakeholder analysis. The stakeholders 7 through 9 will be left out since they are not as relevant to the decision-making processes.

25.3.4   Stakeholder interests

In this part the interests of the different stakeholders in Terraform are identified. A table containing the interests per stakeholder can be found in the appendix B.

---

[3]HashiCorp. "About Hashicorp." 2018. https://www.hashicorp.com/.

The main conclusion is that it is important for HashiCorp and the core team to correctly identify what it is their users want. Different companies (AWS, GCP, Azure, etc. but also Chef, Puppet, etc.) want to work with them and combine their products to create cool solutions. However, HashiCorp needs to focus on what it is their users want and spend their resources on making that happen rather than satisfying these other stakeholders, since that is their own goal. This also aligns with their vision of being cloud agnostic.

25.3.5 Power-interest grid

In the following diagram we show the power and the interest the different stakeholder have in the project.
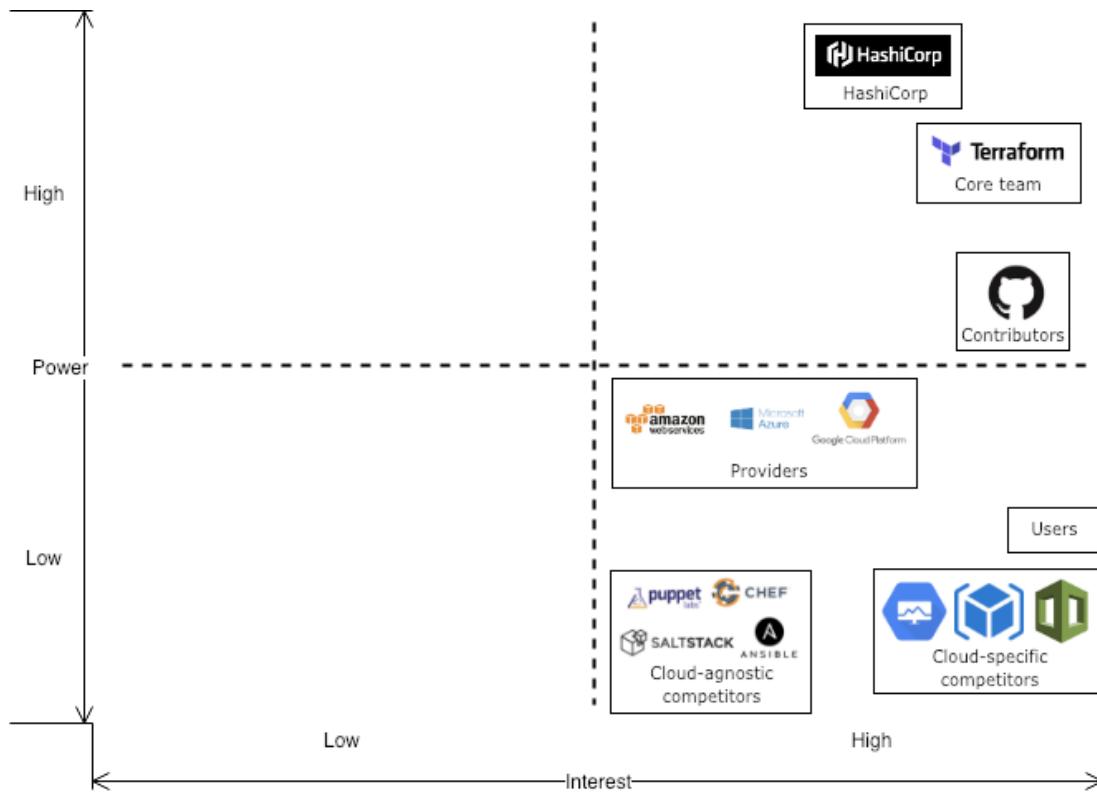
Figure 25.1: power-interest

It can be concluded that HashiCorp and the Terraform team have a lot of power in the project. So even though different stakeholders might put pressure on them to move in their direction, they have enough power to determine their path themselves.

After identifying the interests of the involved organization involved we try to identify people who actually make the day-to-day decisions and look at how they argue their decisions. After this we identify interesting people to contact.

25.3.6   Identifying integrators

Members of the core team of Terraform are also the integrators on the GitHub project. Some of the most active integrators at the moment (March 2019) are the following: apparentlymart, jbardin, radeksimko, paddycarver, mitchellh, catsby, stack72 and svanharmelen. Whenever contributors fix an issue or bug they quickly check and confirm that it works and merge the pull request, however when they find a contributor adds a feature that would change workflows or requires decision-making in a different way they add a `thinking` label and comment that this will be discussed in the team. They write a comment mentioning the reasons why this requires further discussion. Reasons often mentioned are the following:

- Usability
- Semantics
- Backward compatibility
- Consistency
- Security
- Proper testing (acceptance tests are mandatory)
- Proper documentation

We can recognize the user-centered approach we identified before in this reasoning.

25.3.6.1   Pull request analysis

To derive a general theory on the way contributions evolve and are discussed, pull request containing important features, long discussion or that were created by Hashicorp themselves were analysed.

There's notably a strong community, as the bigger part of PRs are finished, tested thoroughly and merged. There are some cases were an author starts working on new features, implements them to the point it is usable for their own use case but neglects to finish it to the point it works in all possible cases, or to adhere to the requested quality standards. Usually however these partial implementations are picked up by someone else in a new branch.

The general flow of a contribution starts with a user already having implemented most of the new functionality. When the architecture is not in line with Terraform, a Hashicorp employee insists on improvements. When cloud provider services do not model nicely to Terraform, authors propose an implementation and the community and Hashicorp discuss the best way to move forward. Acceptance tests and documentation updates are required. The Hashicorp reviewer usually waits before some community members have done their checks, after which they check themselves. There are occasions this takes a really long time, some requests have waited more than 3 months for a review.

Sometimes a small bug is ignored in favour of adding the new functionality, and a new issue is generated for this bug to be solved after merging.

25.4 Context View

25.4.1 System scope and responsibilities

Terraform is an open source project by HashiCorp. It is an Infrastructure as Code (IaC) tool; with Terraform it is possible to manage and provision IT infrastructure without having to configure physical hardware. Using IaC, the infrastructure can be managed under version control. This allows comparison of configurations, e.g. for discovering bugs. Aside from version control, it also allows for consistent deployments [4].

The responsibilities of Terraform are to describe a certain infrastructure and the desired configurations for it. With Terraform, the user can more easily see which changes were made in the configuration. The program knows what has changed and can create incremental execution plans [5].

25.4.2 Context model

The figure shows a context model of entities Terraform deals with in some way. A short explanation will be given for the less trivial relations [6] [7]:

- **Development:** Terraform is developed by Hashicorp, an IT-infrastructure company. Collaboration and version control are handled via Github, where the code of the project (95% Golang) is stored.
- **Communication:** Developers of Terraform use Gitter to communicate. Furthermore, there is communication via Facebook, Twitter and YouTube videos. Terraform also uses MeetUp for groups in various countries.
- **Provision:** Terraform helps creating and managing infrastructure, and also in provisioning during creation or deletion of resources. Provisioners, such as chef, are used to execute scripts (locally or remotely) as a part of resource creation or destruction.
- **Facilitation:** Terraform facilitates many providers such as DNSimple, AWS etc. These are generally Infrastructure, Platform or Software as a Service. The providers in the model a just a few of the many that Terraform facilitates.
- **Collaboration:** HashiCorp closely collaborates with its cloud partners to ensure Terraform works well with these platforms.
- **Paid services (resellers):** Companies like AHEAD use terraform in services they provide to their customers.
- **Competition:** Interestingly, some of Terraforms partners also have their own Infrastructure as Code services. AWS Cloud Formation, Azure Resource Manager, Google Cloud Deployment Manager offer IaC tooling similar to Terraform. These tools only support their own platform, whereas Terraform is cloud agnostic.

---

[4]HashiCorp. "Terraform Introduction." 2018. https://www.terraform.io/i ntro/index.html.
[5]HashiCorp. "Terraform Introduction." 2018. https://www.terraform.io/i ntro/index.html.
[6]HashiCorp. "HashiCorp Partners." 2019. https://www.hashicorp.com/ partners.
[7]HashiCorp. "Terraform Github." 2019. https://github.com/hashicorp/t erraform.
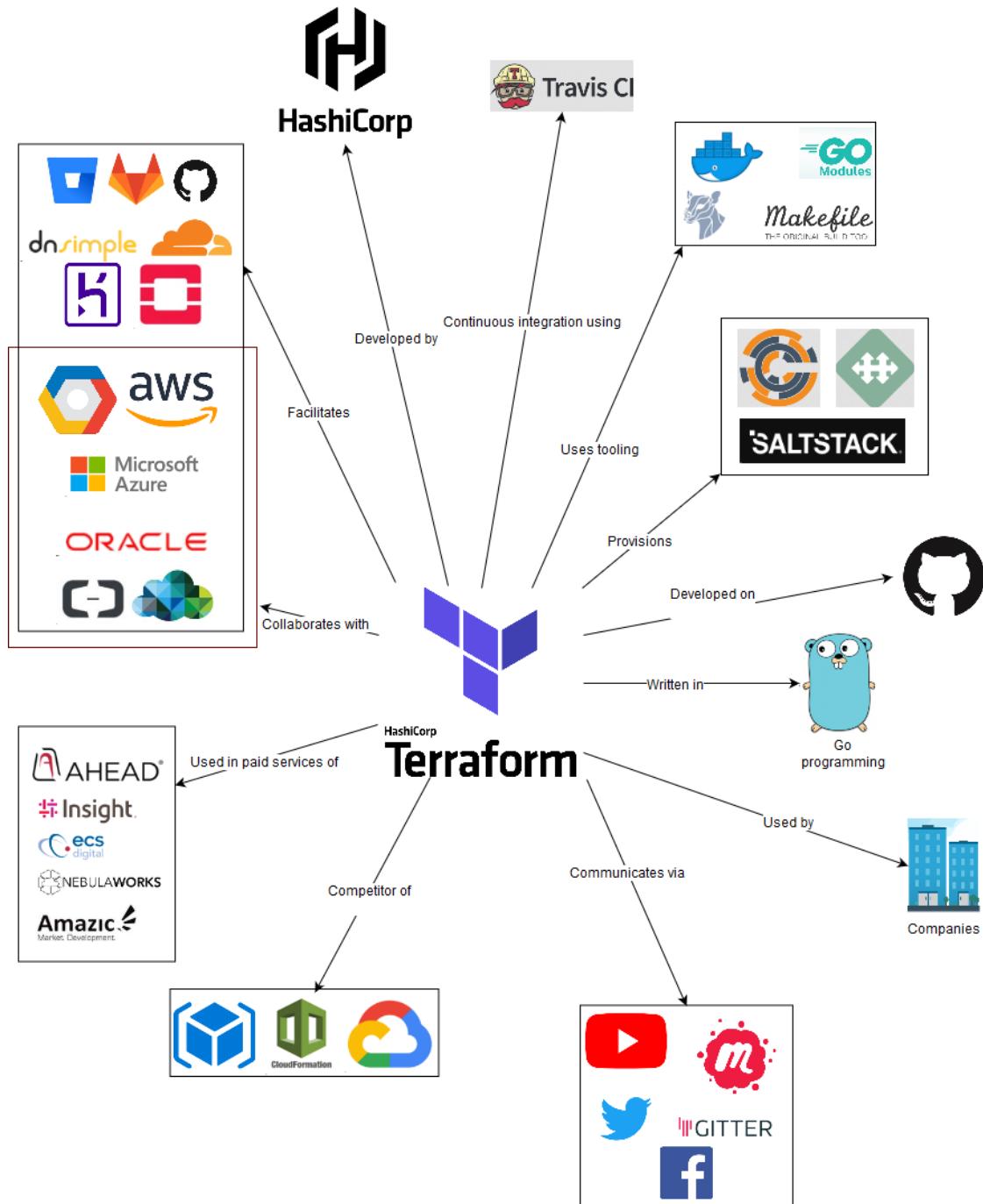
Figure 25.2: Context model illustration

25.5 Development view

The development view describes the system in such a way that stakeholders who build, test, maintain or enhance the system are supported [8]. Below the development view is subdivided into the module composition describing the different modules and their relation to the core, the state of the system, testing methods and the standardized practices of contributing.

25.5.1 Module composition

Terraform is written in Go, where package management by convention is done using a directory structure. Hence, for the following paragraph the top level directories will be denoted as 'modules' while their subdirectories will be denoted as 'packages'. This will make it easier to understand the module organisation of Terraform.

To give a high-level overview of the module organisation, it was oriented around the `terraform/terraform` module, which is often denoted as the core. This provides insight in how the core module links the other modules and packages to provide the Terraform functionality. In the figure below, there are a few modules not directly used by the core package; these modules are coloured red.

Below follows a brief summary of each module, this can be a summary of the `doc.go` that is inside each module or an analysis of the code with respect to the `terraform/terraform` package:

- **terraform (core)**: The core package contains the classes related to the graph traversal (the directed acyclic graph 'roadmap' of Terraform for applying the infrastructure changes to a provider), the different types of nodes that the graph can contain, transforms, user interface logic and evaluations that nodes can perform. These classes combined perform the core logic of Terraform's goal to manage infrastructure.
- **providers / provisioners**: The providers/provisioners module provides an interface of communication to a provider or provisioner during evaluation of nodes in the graph traversal.
- **tfdiags**: Diagnostics module of terraform. Packages can use this module to generate lists of diagnostics which can inform the user of warnings and errors that might occur.
- **version**: Module containing SemVer information
- **addrs**: Module that exposes types that can link to a terraform state or configuration object.
- **helper**: Directory of high-level helper packages for Terraform. Described as the "Terraform standard library" [9]
- **plugin**: Module that is responsible for downloading, installing and discovering locally installed plugins for providers and provisioners.
- **dag**: Module containing the direct acyclic graphs used in the graph traversal.
- **lang**: Module that contains the Go implementation for functions that can be used in the Terraform configuration syntax (e.g. "tomap").
- **plans**: Module that represents the changes that Terraform 'plans' from the previous state and the configuration files.
- **httpclient**: Module that provides a http client with additional configuration such as the Terraform user-agent string.

---

[8]Rozanski, Nick, and Eoin Woods. 2011. Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives. 2nd ed. Addison-Wesley Professional.

[9]mitchellh. "README.md - Terraform/Helper." 2019. https://raw.gith ubusercontent.com/hashicorp/terraform/master/helper/README.md.
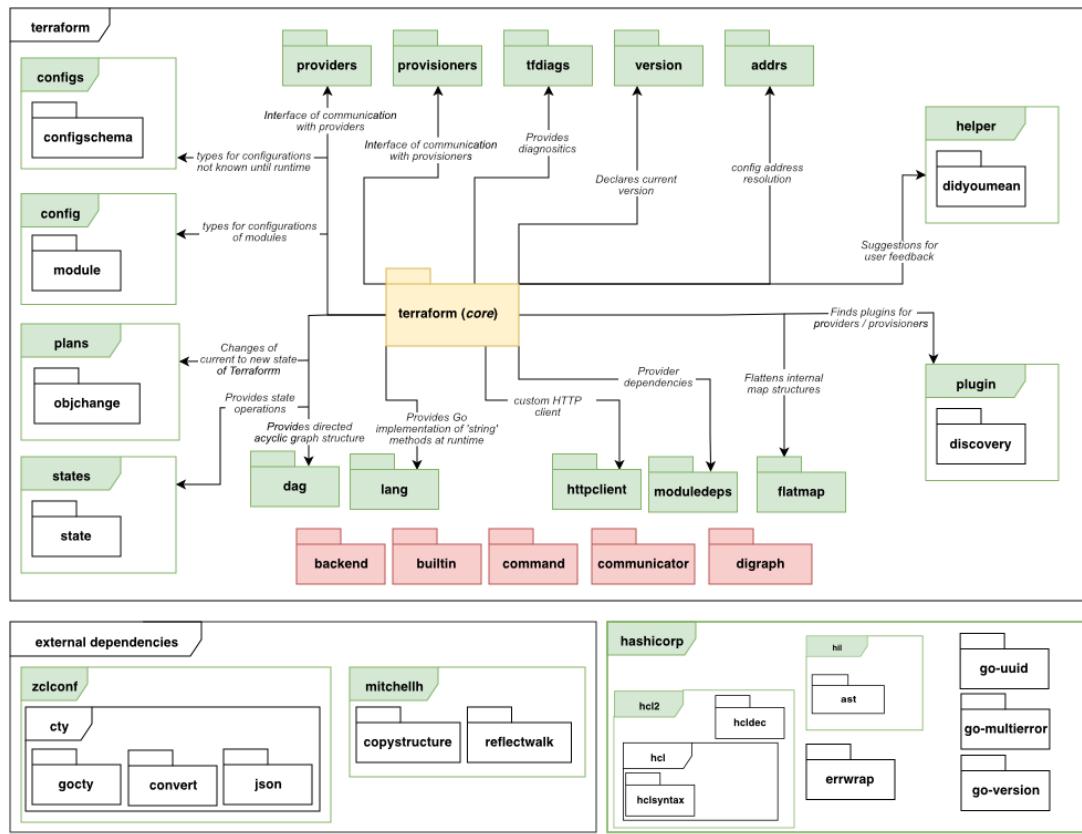
Figure 25.3: module-organisation

- **moduledeps**: Module that exposes types that represent dependencies of a provider or configuration.
- **flatmap**: Module that flattens internal maps.
- **states**: Module that handles state operations and types.
- **config / configs**: Modules responsible for reading the terraform configurations written by the user.

Other modules not directly linked to the terraform core package are:

- **backend**: Module that handles storing the state of Terraform to either local or remote storage.
- **builtin**: Built in provisioners and providers.
- **command**: Module that exposes commands that can be executed (e.g. using CLI)
- **communicator**: Module that exports interface of communication for provisioners.
- **digraph**: Module that exports basic structures for creating directed graphs.

Other (external) entities used by Terraform are:

- **zclconf**: A module that provides a dynamic type system for Go primarily for configuration languages [10]
- **mitchellh**: Github user that exports two Go modules: copystructure [11] and reflectwalk [12]. Copystructure deep copies values while reflectwalk allows manipulations on unknown structures.
- **hcl2**: toolkit for writing structured configuration languages [13]
- **hil**: Module for configuration interpolation originally written in Terraform but extracted as a general purpose module [14]

25.5.2   Providers / Provisioners

The providers and provisioners mechanism is an interesting component to highlight. During the graph traversal Terraform only sees the interface that all providers / provisioners should conform to. The providers start a gRPC server which is compliant with the protobuf files in `internal/tfplugin5`. This creates an 'airgap' between the actual provider (e.g. AWS, Azure, etc..).

25.5.3   State

Terraform stores it's state in a `.tfstate`. Access to the state file is managed through the `statemgr` (state manager). Schemas of the state are stored in the `terraform/states` package. Writing, reading and migrating the actual state file itself is done using the `terraform/states/statefile` package.

25.5.4   Testing and Static Analysis

Terraform uses (by Go convention) files with `*_test.go` to denote tests. Additional data used in the tests are stored in `fixture/*` directories. There is also one package related to `e2e` testing which is conveniently called `e2e`.

---

[10] Atkins, Martin. 2019. "Zclconf/Go-Cty." 2019. https://github.com/zclconf/go-cty.

[11] mitchellh. "Mitchellh/Copystructure." 2019. https://github.com/mitch ellh/copystructure.

[12] mitchellh. "Mitchellh/Reflectwalk." 2019. https://github.com/mitchellh /ref lectwalk.

[13] HashiCorp. 2019. "Hashicorp/Hcl2." 2019. https://github.com/hashicorp/hcl2.

[14] HashiCorp. 2019b. "Hashicorp/Hil." 2019. https://github.com/hashicorp/hil.

25.5.4.1 Static Analysis

The terraform repository code formatting is analysed using `gofmt`. This is integrated in the `Makefile` and checked before running any tests.

25.5.4.2 CI/CD

Travis is used for CI/CD. It uses the `Makefile` and checks if the formatting is correct (prerequisite of `make test`), runs the unit tests and runs the e2e tests. After all tests have completed a `go build` job is executed to test if a binary can be created.

25.5.5 Standardized practices

There are a few guidelines for contributing to Terraform, described on their GitHub page. First of all, one must sign the CLA. Furthermore the community guidelines must be adhered. These cover being respectful in communication, harassment policy and incident handling. Hashicorp remains the right to perform punitive actions.

In contributing to Terraform, the community emphasizes that any sort of contribution is appreciated, the worst that can happen is that you'll politely be asked to change something. A description of what is expected in a feature request, new issue or new pull request is provided. Also a checklist has been put together for contributions per type of contribution (Documentation update, enhancement / bug fix to a resource or provider, a new resource or a new provider). Adhering to this list helps in achieving quick merges. Finally acceptance tests are required for each new or updated feature. All of this is described in detail, including all descriptions and checklist, in appendix C.

25.6 Technical Debt

To analyse Terraform's technical debt, SonarQube, BetterCodeHub and an analysis tool of the GoLand IDE were used. More information on the results of the tools can be found in appendix D.

25.6.1 Types of Technical Debt

There are several types of technical debt, for example: code debt, testing debt, defect debt and documentation debt. It was decided to discuss code debt and testing debt, since documentation debt and defect debt are scarce in the project.

25.6.1.1 Code Debt

According to SonarQube, Terraform v0.12.0-beta1 has 1.4k code smells and v.0.11.0 has 507. The large amount of code smells is bad; it means maintaining the code will be harder than it should be and additions to de code can lead to new errors in the code. The code debt expressed in number of days to fix it, is 29 days

for the beta version and 12 days for v0.11.0. The technical debt for v0.12.0 is a lot and they should reduce this before making an official release.

BetterCodeHub found a fair amount of classes do not have short units of code written in them. The Lines-of-Code (LoC) count for functions of the worst offenders ranges between 111 and 193. The tool also checked how simple units of code were. The results were bad, manual checks in the code revealed extremely long functions filled with many if-statements. The two worst functions have over 130 LoC and over 30 branches. As was also seen in the SonarQube analysis, the code also has some duplication. Several classes have between 13 and 18 LoC duplicated. Lastly, unit interfaces were not kept small, the worst functions had 5 parameters.

25.6.1.2 SOLID

The code was also checked manually to see if there are violations of the SOLID design principle. Since the project is so large, we focused on the terraform core package. The *single responsibility* principle was met. For example, the package structure is very extensive; this is because all different responsibilities are split over different packages. We also saw various structures that had very specific tasks such as 'destroy edge'. The *open-closed principle* was met in certain instances. A good example is the refactoring of the providers/provisioners in which they removed much duplicate code and replaced it with a plugin system containing an interface. This is open for extension but closed for modification. However, in other structures we saw many complicated switch and if-structures, which can be an indicator of bad design according to this principle. We found that, concerning the *dependency inversion principle*, Terraform does quite well. They have interfaces all subtypes must comply with.

25.6.1.3 Testing Debt

SonarQube gave a test coverage of 59.8% on the current master branch. However, these numbers do not accurately reflect how well the system is tested, because Terraform uses a unique testing system, some packages were left out as the SonarGo could not process these results. When manually looking at the code, it can be estimated the coverage is a lot higher. Terraform has unit tests for most structures and an end-to-end test structure that takes commands. There has been quite extensive testing and many test fixtures were written. We did find 25 functions with a testing TODO in the core package. These were all in their evaluate modules.

25.6.2 Discussion about Technical Debt

With GoLand's TODO utility, the code was checked for remarks in comments that indicate code needs to be refactored or iterated upon. The terraform/terraform package had the most markers: 61. As of 17-3-19, the code has 157 open TODOS. The developers communicate in pull requests on GitHub as well. When examining some pull request, no direct mention to technical debt was found. However, many pull requests do relate to fixing bugs, testing or code improvements.

25.6.3   Historical Analysis Technical Debt

The history of the technical debt will be discussed by going into the evolution of Terraform's modules and the evolution of technical debt.

25.6.3.1   Evolution of Modules

Terraform is split into modules according to Go conventions. We looked at both the evolution of the LoC and of the file count. The metrics showed exactly the same trends, so it was decided to include the figure of the evolution of the LoC shown below.
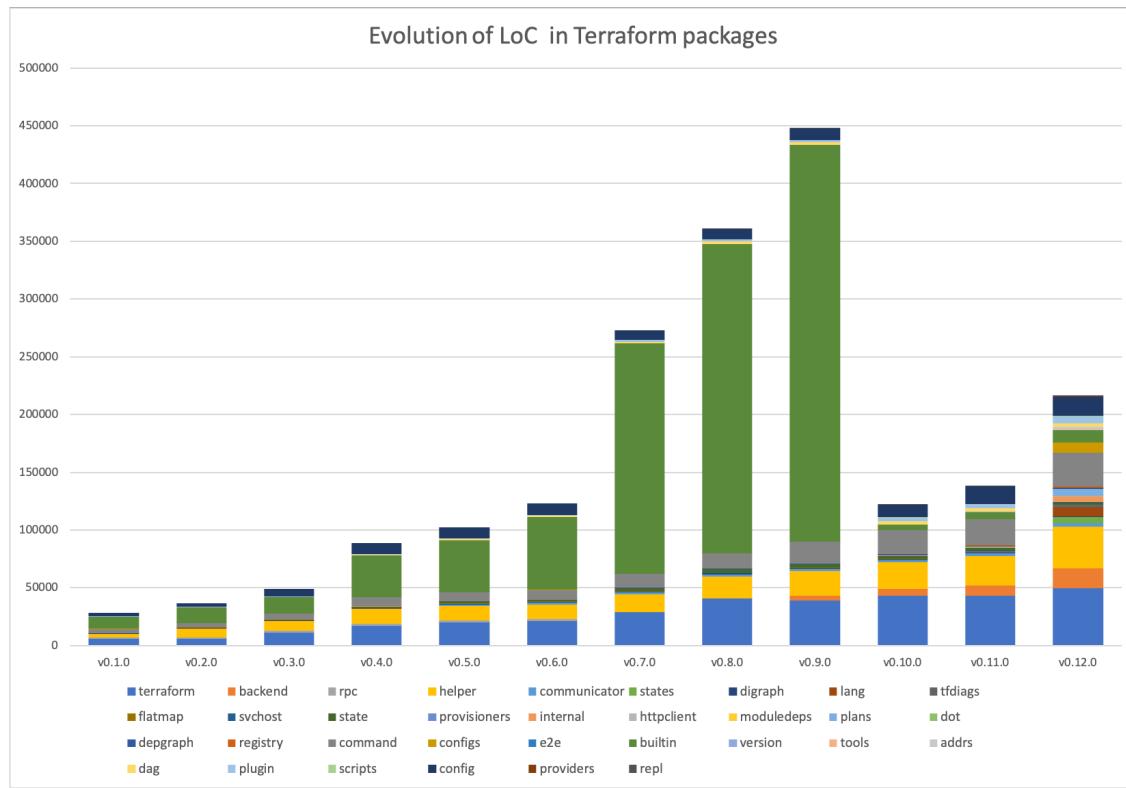
Figure 25.4: Evolution LoC

The evolution of software components can be described using astronomical terminology as described in the paper written by [15]. The evolution of a few modules will be described in more detail using this methodology:

- **digraph**: this is a typical example of an *idle* module with good design. The module has not been touched since v0.2.0 while still being used in the project.

---

[15]Lanza, Michele, and Stéphane Ducasse. "Understanding Software Evolution Using a Combination of Software Visualization and Software Metrics." Springer Verlag, 2002.

- **command**: this handles the commands which Terraform can execute. As Terraform grows in functionality this module grows. This module has been growing since v0.1.0. This means it classifies as a *red giant* and might need to be refactored at some point.
- **builtin**: this module could first be seen as a *red giant* until the refactoring in v0.10.0. However, after this big refactoring the class is growing in size. Which means we can classify this module as *pulsar*.
- **state**: this is another example of a *pulsar* module as its size has been varying since the launch of Terraform. An argument can be made to classify this as a *white dwarf* since it has been slowly shrinking since v0.8.0
- **configs**: this module has remained constant (as an idle, or even sleeper, module) from v0.1.0 to v0.11.0, however due to the refactoring and launching of the plugin system this module suddenly exploded to contain many more configurations. Therefor this can be classified as a *supernova*.

25.6.3.2 Evolution of Technical Debt

We combine this data with the metrics we gathered by using SonarQube for different releases. The diagram below shows the technical debt in days of work over the last few releases. It was not possible to run SonarQube for versions before v0.7.0 as dependencies were no longer available.
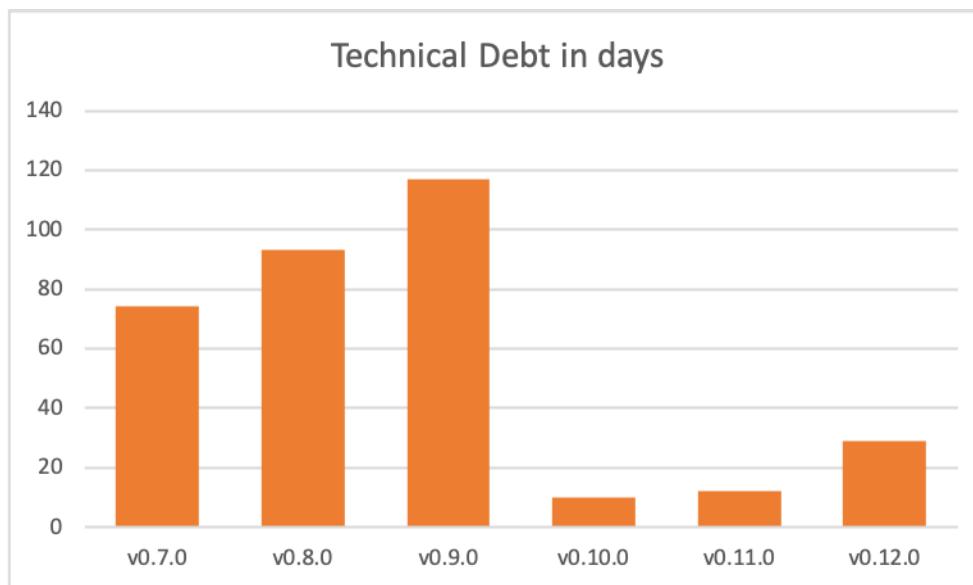
Figure 25.5: Technical debt in days

In version v0.9.0, 1925 of the 2045 code block duplications are in the terraform/builtin/providers package. This package was moved to github.com/terraform-providers in v0.10.0 and in v0.12.0 moved to the new plugin system. This also accounts for 109d of the debt and therefore greatly decreases the technical debt. They did introduce some technical debt in this migration by postponing the implementation of the plugin system to v0.12.0 instead of directly implementing this. The introduction of many new modules over the last couple of versions is a promising sign that the core team tries to manage technical debt as the system grows larger.

25.7 Usability Perspective

In this chapter the usability of Terraform will be discussed. In order to gain insights in how users experience working we Terraform the team interviewed an enthusiastic developer who recently introduced his company to Terraform and is working with it in a professional environment.

25.7.1 User Interaction

User interaction is an important part of the usability of the software. Great software can have a very non-intuitive user interface and therefore be of no good to anyone. In this section, the interface of terraform is analysed. Terraform uses a command line interface, which will be described below.

25.7.1.1 Command Line Interface

Terraform is used via a CLI. This CLI should be installed and can be executed in any directory containing `*.tf` (Terraform configuration) files. The commands contain a synopsis which can be found by executing `terraform` or `terraform --help`. Every command also contains a larger documentation using `terraform <command> --help`, which will output a larger summary and possible arguments that can be supplied to the command.

The most common commands to use are `terraform init` (which initializes a new or existing directory), `terraform plan` (which generates an execution plan for later use) and `terraform apply` (which applies the current configuration to a provider).

25.7.2 Learnability

How easy is it to learn Terraform? That depends on what would be considered 'learning Terraform'. According to the interviewee, learning to write Terraform's configuration language is not that difficult. However, one could also consider learning about infrastructure and cloud services of specific providers as part of learning Terraform. Including this makes the learning curve steeper. Infrastructure can become very complicated, especially as it gets larger. Cloud services of specific providers can be very complex, and working with Terraform to create infrastructure on them does require knowledge of how they work. Also infrastructure coded with Terraform for provider x, is not interchangeable with code for similar infrastructure on provider y.

25.7.3 Effectiveness

The goal of Terraform is to provide an easy, cloud-agnostic way to write infrastructure as code (IaC). This allows infrastructure to be managed and deployed more consistently and allows for faster provisioning of resources. Also, software development best practice (e.g. use of Git) can be applied to infrastructure and its development can be viewed over time. These benefits will differ between companies, e.g. companies which have to spin up the same infrastructure in different situations (for example to create systems for their clients) will experience way bigger advantages compared to a company which just has to host their relatively simple webserver.

25.7.3.1 Migration to Terraform

Migrating to Terraform requires at least some employees in an organization to familiarize themselves with the tool as well as with IaC as a concept. In the previous paragraph it was concluded that it is not necessarily very difficult to learn how to use Terraform, however this does take up time which might not be available. Our interviewee estimated this process to take up to a week, however he did have a lot of experience with infrastructure. If having no experience with infrastructure or infrastructure is not an important part of a company, learning Terraform may not be a good investment. After learning the tool existing infrastructure can be imported easily, and the tool is quickly setup.

25.7.3.2 Overhead in using Terraform

Because Terraform allows infrastructure as code, the infrastructure is often managed under version control. A branch structure and pull requests are often used to collaborate. Along with the benefits this has, it also introduces some overhead to create all the documentation around the code and changes to the code. Besides this Terraform does not provide any real overhead.

25.7.3.3 Reliability of Terraform

A major concern in the decision to use Terraform is the reliability of the tool. A created configuration can be applied many times in exactly the same manner, which is a reliable feature. However, a reliability risk of Terraform is a state called drift, which is a term for when the real-world state of the infrastructures differs from the configuration. This cannot be detected automatically, and must be fixed by manually updating the configuration or infrastructure. Being aware of this responsibility is important when working with any IaC tool.

Furthermore, Terraform is still quite young: a v1 has not yet been released, and new features are being added frequently. On the other hand, some big corporations are already using Terraform in large projects so some battle testing has been done.

25.7.4 Collaboration

One of the biggest selling points of Terraform is being able to work on an infrastructure configuration via version control with a team of people. This can be done rather easily, as long as the state is stored remotely. A local state makes collaborating hard, as it should be checked into and pulled from version control with each change.

The interviewee explained that currently their CI/CD pipeline is in development. At the moment they do code review via PRs on GitHub, and once merged manually apply the configuration. Automatic unit testing could be done using TerraTest.

25.7.4.1 Contribute to existing projects

Joining in development of an existing Terraform should not be hard, as long as the configuration has been set up modularly. This means certain blocks of responsibility in the infrastructure configuration have been

separated. The Terraform files are self-explanatory for the most part, assuming knowledge of the syntax and the way cloud infrastructure is orchestrated.

Terraform supplies a `graph` command, which combined with for example GraphViz's `dot` command can be used to draw a diagram of the Terraform resources according to the configuration files in the current directory. This visualization can help in getting a feeling for the existing configuration.

```
$ terraform graph | dot -Tsvg > graph.svg
```

25.8   Conclusion

Terraform is an interesting Infrastructure-as-Code project with many benefits, e.g. version control in IT infrastructure management. With the availability of many cloud services, this type of software can become very important in the future.

We identified the stakeholders and concluded the power lies mainly at HashiCorp and the Terraform core team. Providers, such as Amazon and Azure, can have an interest, and a fair amount of power. For example, if an API for a service is not made public yet, Terraform may not be able to call it. We denoted this as them being on the high side of low power, as this is more of an indirect influence.

In the second chapter, we looked at Terraform in its context. It is written in Go language and uses Fossa and Travis CI for continuous integration. We saw it is not only used by companies for their infrastructure, but also by resellers who teach Terraform or configure infrastructure for companies. Terraform collaborates with various cloud providers and facilitates a few more. It also competes with a few, like CloudFormation by Amazon. However, it is not real competition as such; these cloud providers will still make money when people use their resources with Terraform.

In the development view, we looked at interesting details for developers wanting to work with Terraform. The module structure of Terraform is mostly oriented around the terraform core package. Among these packages are the provisioners/providers, tfdiags (diagnostics) and plugin. In addition to this, it has packages like the backend (stores Terraform state) that are not directly linked. Lastly, Terraform has a few external dependencies such as Hcl2, a toolkit for writing structured configuration languages. We also looked at guidelines for contributing to Terraform. The developer must sign the CLA and adhere to the community guidelines. Acceptance tests are required for each new or updated feature.

Looking into Terraform's technical debt, we found much duplicate code and many code smells. The technical debt is 29 days for the beta version and 12 for v.0.11.0. It also has very long functions with many if-statements. The testing debt seemed great, but the tools did not accurately reflect all the tests. Looking through them manually, there has been quite extensive testing. An interesting discovery we made in the history of technical debt was that v0.10.0 suddenly had a drastic decrease in technical debt (> 100 days). This was because Terraform moved to a new plugin system, but it is debatable whether this is a true decrease.

Lastly, we introduce a usability perspective. We looked at what Terraform's UI is like; it uses a command line interface. Terraform appears quite easy to learn. However, it does require knowledge about infrastructure and cloud services, which makes it more difficult. Terraform can be effective as it allows for version control and faster provisioning of resources. A reliability risk of terraform is a state called drift; when the real-world infrastructure differs from the configuration. This cannot be detected automatically and must be fixed manually. A benefit of Terraform is cooperating on infrastructure via version control. This is easy when the state is stored remotely, a local state makes this almost impossible.

25.9 References

References in footnotes.

Figure 25.6: Chapter_header

Chapter 26

Vim

Vim is a text editor most commonly used by system administrators who use it as a command line tool to edit scripts, texts and code. Vim is highly customizable with multiple shortcuts and commands available. It is this flexibility that has built its popularity but also the fact that Vim is open source and therefore users have a big say in the development of the product. Vim was originally created by Bram Moolenaar in an attempt to recreate an editor he had previously used, VI, and hence the name was born, "VI Improved", what became Vim in short. The first release of Vim was in November 1991 making it older than most students in this course. Vim quickly established itself as a stable and reliable editor and became the editor-of-choice for system administrators. Despite the arrival of commercial editors such as Microsoft VS, Notepad++ and many others, Vim still remains popular, especially with the aforementioned system administrators. Bram still remains the main developer of Vim and due to a broad and active community of developers, updates are still common with around 30 commits per week on average and a lively mail list discussing potential changes and improvements.

26.1   Table of content

26.2   1. Stakeholder analysis

A stakeholder in the architecture of a system is an individual, team, organization, or classes thereof, having an interest in the realization of the system [1]. This paragraph describes the different stakeholders in the Vim project. The types of stakeholders are based on types defined in the book of Rozanski and Woods [1]. Next to those types, three more types of stakeholders have been defined, namely integrator, sponors and eco-system enhancers.

26.2.1   1.1 Core developers

26.2.1.1   Bram Moolenaar (@brammool)

Stakeholder type: **Acquirer, Developer, Maintainer, Production engineer, Communicator, Integrator**
Bram is the author of the open-source project Vim and is actively updating the runtime files for maintenance purposes. The authorship defined him as the acquirer of this project. More importantly, he is in charge of adding bug fixes over OS compatibility, platform compatibility and functional inconsistencies, thus he serve as the maintainer as well. This developer also decides the future roadmap of the development of Vim. Bram is the integrator, meaning he is the one who in the end decides to merge a pull request (PR) with the master branch [2].

26.2.1.2   Henk Elbers, Eric Fischer, Dany St-Amant, Roger Knobbe & more

Stakeholder type: **Developers, Maintainers, Production engineers**
This list of authors (non-exhaustive) lists authors that are production engineers and developers that help porting Vim from a single platform to Windows, Mac OS and various other operating platforms [3].

26.2.2 1.2 Other developers

26.2.2.1 K. Takata & Christian Brabandt (@k-takata, @chrisbra)

Stakeholder type: **Assessors, Testers, Developers, Production Engineers, contact person**
K. Takata and Christian Brabandt are reviewers for potential merge requests that add functions to Vim or attempt to fix bugs, thus defined as assessors. They are also members of Vim repository and Vim Github base. They are also the contact person for user-oriented and developer-oriented queries.

26.2.2.2 Enno (@konfekt)

Stakeholder type: **Developer, Tester, Contributor**
Enno developed several plugins for Vim and is actively contributing to the core Vim repository by proposing new features and bug fixes, thus named as testers and contributor.

26.2.2.3 Dominique Pelle (@dpelle)

Stakeholder type: **Developer, Assessor, Contributor**
Dominique actively contributes to the main Vim codebase by reviewing pull requests, proposing new features and bug fixes and producing valgrind reports for Vim [4].

26.2.3 1.3 Communicators

26.2.3.1 Dan Sharp and Stefan 'Sec' Zehl

Stakeholder type: **Communicators**
Dan is the active maintainer of the official Vim documentation on Sourceforge. Stefan is the maintainer of the official website of Vim open-source project.

26.2.4 1.4 Communities

Vim users, developers and sponsors formed various Vim communities that actively contribute to both the administrative and program development of the Vim project.

26.2.4.1 Vim Github Repository members

Stakeholder type: **Support Staff, Developers, Maintainers**
The Vim Github Repository is very active with around 30 posts per day. Github Repository members are users that comment on issues, PRs and assist each other with issues and problems that come up. The same persons also propose new features and submit PRs for other members to evaluate.

26.2.4.2   Vim Plugin Developers

Stakeholder type: **Eco-system enhancer**
Vim Eco-system enhancers are a group of programmers who actively develop and maintain Vim plugins. A plethora of Vim plugins has been created allowing users to customize their own Vim experience. Some of those plugin developers are also involved in the development of Vim itself, but all of them depend on the Vim source code being stable and error free.

26.2.4.3   Sponsors

Stakeholder type: **Sponsor**
Sponsors (list of Vim sponsor here) are a group of donators who financially support Vim's development by donation and they are crucial for the project [5], all the money donated goes to a charity which has been selected by Bram personally and donations increase his and others' motivation to continue their efforts on Vim [6].

26.2.4.4   Vim users

Stakeholder type: **User, Tester (if applicable), System Administrator**
Vim users use Vim and its components from basic word processing, development operation, software development and administrations. Vim has a relatively large marketshare in developer communities and integrated development environment market. Vim constitutes diverse user types, ranging from students and code learners with a relatively higher learning curve due to the non-graphical user interface and keyboard shortcuts. Expert users are adjusted and acquainted with the editing and programming style based on their adaptation to the Vim editor.

26.2.4.5   Google help group

Stakeholder type: **Support Staff**
A google help group is a collection of users and their discussions on Vim-related issues. Users are encouraged to ask for help in a Google group. The Google group consists of several mailing lists, of which one is specifically for discussion on the current development of Vim. Several of the individuals mentioned earlier are present and active in that group, however so are other individuals who have not been mentioned and therefore this is considered its own group of stakeholders.

26.2.5   1.5 Competitors

Vim competitors are every text editor or IDE, including Atom, gedit, notepad++ and VS code. However, the main competitors of Vim are the ones in the same niche - command-line text editors including EMACS and Nano: * MACS is an extensible, customizable text editor. At its core, Emacs is a Lisp interpreter that just so happens to support text editing. However, it includes a bunch of plugins that greatly extend its functionality. * Nano is a text editor for Unix-like computing systems or operating environments using a command line interface. It is licensed under the GNU General Public License (GPL), thus it is very popular to use in command-line.

26.2.6   1.6 Managing stakeholders

In each project stakeholders have to be managed. To get an overview of the power and interest each stakeholder has on the realization of Vim a Power-Interest grid can be found in figure 1.
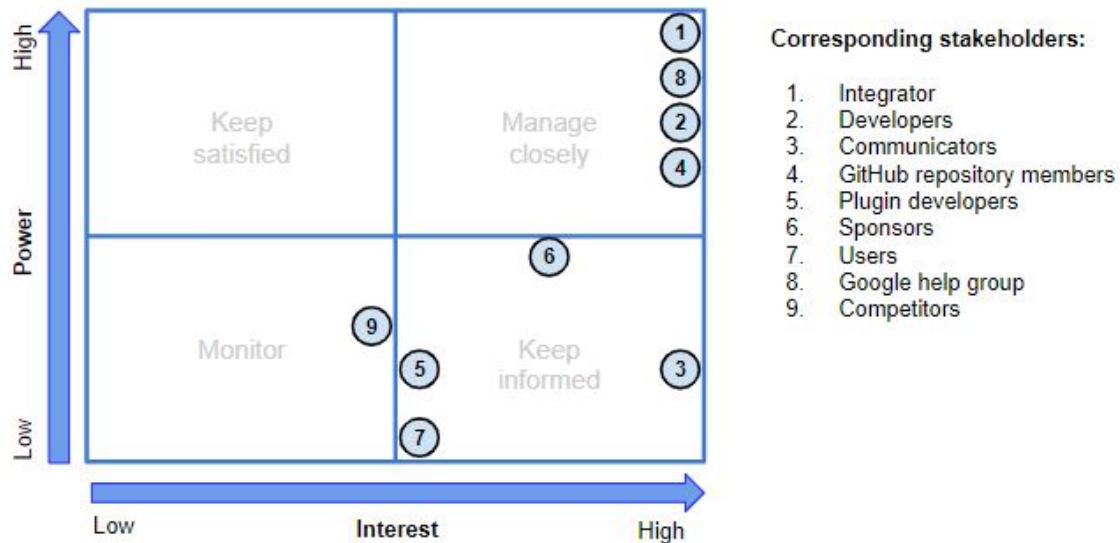
Figure 26.1: PI_grid

Figure 1. Power-Interest diagram for Vim project

The integrator, developers, communicators and GitHub repository members all have very high interest in the realization of Vim. However, there is only one integrator and thus he has the highest amount of power. The individual developers also have a very high amount of power, but lower than the integrator. The less active developers, represented by the GitHub repository members, have less power than the active developers. The communicators have almost no power on the realization of Vim, but do have some power since they communicate version release, updates about the development of Vim, etc. The Google help group has a lot of power, because PRs are discussed here before they are made on GitHub and the Google help group consist of highly involved developers and thus it has high interest in the realization of Vim.

The plugin developers do not have direct power on the realization of Vim, but also do have some power as plugins positively influence Vim competitive position. The competitors have more power than the plugin developers, because they can influence the market share of Vim and competitor's developments can incite Vim's developers to alter Vim. Competitors need to be monitored and plugin developers need to be kept informed, as they might need to alter the code of their plugins if Vim's code changes.

Sponsors donate money to Vim, which eventually goes to a charity. They donate money to stimulate the realization of Vim and are thus interested in the system. Since the donations motivate the (core) developers, they have some power as well.

Users have almost no power in the realization of Vim, but users stimulate the developers to work on Vim and thus have a little power. Users do have I high interest though, because they depend on Vim.

26.3   2. Context view

In this paragraph we are going to describe relationships, dependencies, and interactions between the system (Vim project), its environment and the ecosystem in general. Firstly, the system scope is described, followed by the context model scheme and a detailed explanation of relationships with external entities.

26.3.1   2.1 System scope and responsibilities

Vim is known as a stable editor and it is continuously developed as an open-source project. Its prominent features are listed below: * persistent, multi-level undo tree * extensive plugin platform * support for diverse programming languages and file formats * powerful search and replace features * integrates with many tools and platforms

26.3.2   2.2 Context model

Figure 2 graphically displays the context view. The different components of the context view are explained below.
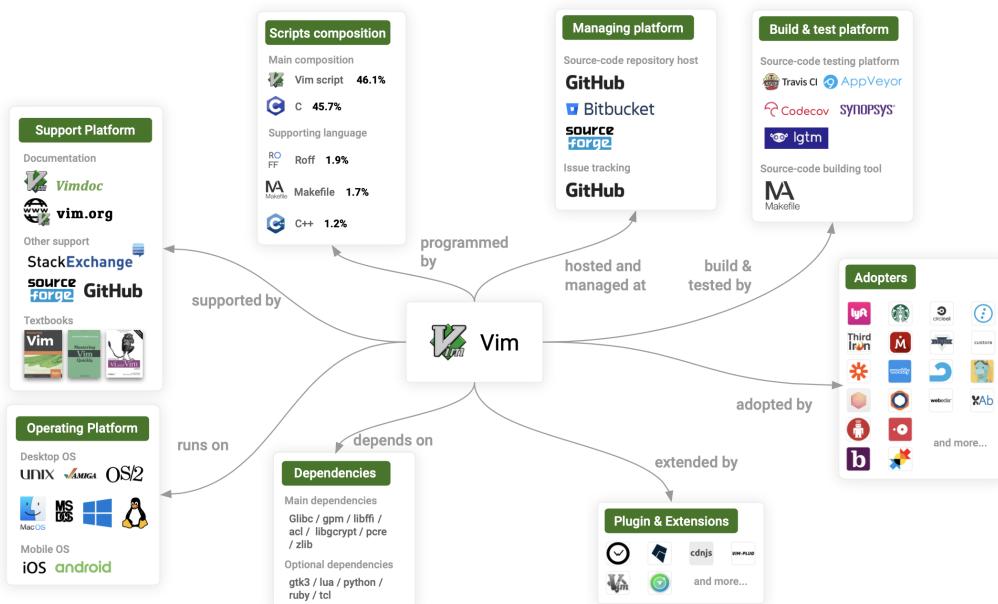
Figure 26.2: context_view

Figure 2. Context view for Vim project

26.3.2.1 Support platform

As a text editor that has such a long history, Vim contains an extensive support platform. Most of the documentation and learning material can be found in vimdoc and on the vim homepage. However, information can also be found on Sourceforge, StackExchange, Github and even in books.

26.3.2.2 Scripts composition

The Vim project is build with a diverse array of scripting and programming languages. The main coding languages are Vim scripts and the C programming language, and the rest is supported by ROFF, Makefile and C++.

26.3.2.3 Managing platform

Vim is managed on three code repository platforms: Github, Sourceforge and Bitbucket. Github is used as a main platform as it also provides issue tracking and is popular in the developer community [7]. Bitbucket repository is used as a back-up repository and for those who prefer Bitbucket over Github.

26.3.2.4 Build & test platform

Vim uses Travis CI and AppVeyor as continuous integration tools (different ones for different platforms). It also uses different tools that help to ensure code quality such as CodeCov, Synopsis and Lgtm. The testing platforms are integrated into Github workflow and test the code continuously and automatically.

26.3.2.5 Operating platforms

Vim is originally written for text editing purposes in the Amiga operating system. With the increasing adoption and population of the tool, Vim has been ported to Windows, MsDos, Mac OS, OS/2 and Unix platforms. Additionally, Vim is available to use even on mobile operating systems including iOS and Android. Operating system is an external entity since it is the environment for Vim to build, install and run.

26.3.2.6 Software dependencies

Vim is composed mainly in C and Vim script, thus there are basic dependencies that need to be implemented to ensure the operating stability of Vim editor. On the other hand, some advanced features of Vim require other dependencies to be implemented to support their functionality. Dependencies are external entity since it provides support and compatibility service to Vim users. Table 1 provides a list of dependencies required or optional for Vim.

Dependency name	Description	Required or optional
Glibc	A standard library for the allocation of C-programming language to ensure Vim's compatibility with hardware devices	Required
gpm	A mouse server for the command-line environment	Required
libffi	A portable foreign function interface library to reduce time for calling natively compiled functions	Required
acl	An access control list utilities, libraries and headers	Required
libgcrypt	A general cryptographic library based on GnuPG code	Required
pcre	A library to implement Perl-style regular expressions	Required
zlib	A compression library that implements deflate compression methods found in gzip	Required
gtk3	A GObject-based cross-platform graphic user interface toolkit	Optional
lua	A powerful light-weight programming and scripting language designed for application extension	Optional
python	A high-level scripting and programming language to support advanced Vim functionalities	Optional
ruby	An object-oriented programming and scripting language for fast and intuitive code development	Optional
tcl	A high-level, general-purpose scripting language	Optional

Table 1. List of dependencies for Vim

26.3.2.7 Plugins and extensions

Vim can be extended by extensions and plugins created externally by people who are not in Vim's development team. These extensions and plugins enhance, add or alter Vim's ability to edit, code, compile and organize files. These extensions include Vim-plug, SpaceVim, NeoBundle and more.

26.3.2.8 Adopters

Vim has been adopted by many companies and organizations due to its popularity in the development circle. Adopters use Vim to edit, compile or organize their coding and ext-based files, and promote the tool to internal employees and managerial staff. The adopters include Lyft, Starbucks, CircleCI and more. Adopters are an external entity since it provides information data and user feedbacks to the developers.

26.4   3. Merge decision strategy

To get an insight into when pull requests (PRs) are accepted or rejected, multiple PRs have been analyzed. Based on this insight, a theory about the merge decision strategy has been formed.

26.4.1   3.1 Criteria for pull requests to be analyzed

The PRs that have been chosen to be analyzed are relatively new, as they are all proposed in the last two months. Vim has been around for a long time and since we are interested in the current merge decision strategy it has been decided to look at relatively new PRs. Other criteria to decide whether a PR would be analyzed and included is the amount of comments on the PR and the amount of new information about the merge decision strategy it would offer. To get a complete picture about the merge decision strategy, approximately the same amount of accepted and rejected pull request have been analyzed.

26.4.2   3.2 Information obtained from pull requests

Often a PR gets accepted by Bram, the integrator, without any explanation and sometimes he comments "Thanks, I'll include it". In a second situation multiple people have a discussion about the PR and decide either that it can be merged or that the proposer has to alter the PR or someone else has to offer an alternative PR, and then it can be included. Sometimes Bram is included in those discussions and sometimes he isn't. In a third situation Bram himself is not certain on whether to merge the PR and asks others to comment on the quality of the PR. The fourth possibility for a PR to be accepted is when a PR is proposed and Bram himself makes an alteration to the PR and includes that.

Of the closed PRs, not a lot are rejected. They might get rejected at first, but if the mentioned alterations are made, it will be merged. If a PR is rejected, often a discussion between multiple people has taken place. In other situations, someone other than Bram comments on a PR and the proposer agrees with the comment and thus closes the PR himself without it being merged. In one situation Bram commented on a PR that he did not see any added value in merging the PR and thus he did not do so.

One of the mailing lists used is specifically for discussion about the current development of Vim, therefore often a lot of discussion takes place before a PR is made. This might be a reason why a lot of PRs get accepted.

26.4.3   3.3 Theory about the merge decision strategy

As there is only one integrator, Bram, he has the final say on whether to merge a PR. But as is stated above, he does listen to other contributors and even asks for input on whether to include a PR. If multiple people see the PR as added value for Vim, whether the change is small or of a larger size, the PR will be merged.

In Appendix A a complete overview of the PRs analyzed can be found, including the information obtained from each of them.

26.5   4. Development view
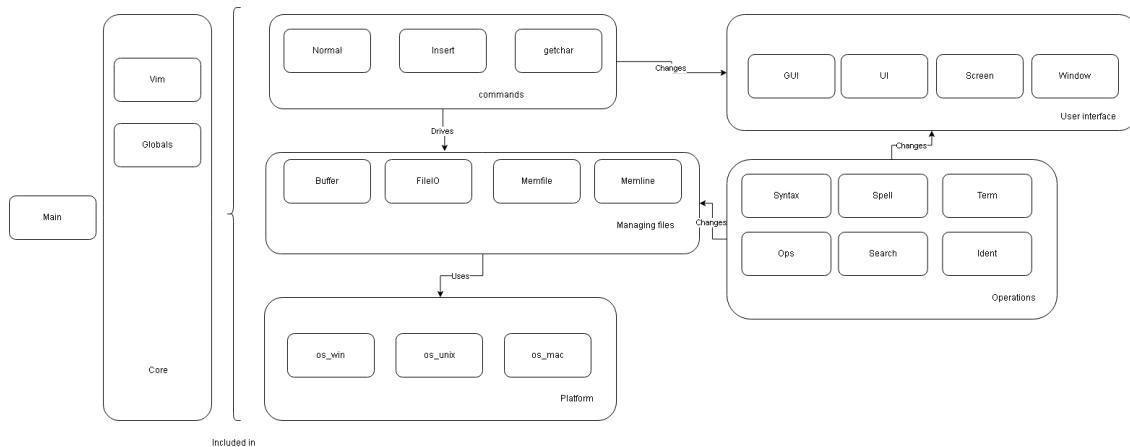
The development view describes the architecture that supports the software development process [1]. First the module structure will be discussed, second the common design model and third the codeline model.

26.5.1   4.1 Module structure model

In this section a module structure model, which shows the organization of the source files into modules that contain related code [1], is discussed. Such a structure provides an overview of the source code which guides developers to understand and navigate the codebase.

Before diving into the module structure diagram several points should be covered: - The Vim codebase is old, it contains more than 26 years of code including updates, patches and continuously evolving functionality. However, the code has never been seriously refactored and is considered to be stable by it's author and core developers. - Most of Vim is written in C, which is a low level language. Thus, it is quite hard to follow the structure of the code.

In figure 3 the core components of Vim may be observed. Each small rectangle represents a logical part of the software which mostly corresponds to a `*.c` file with a corresponding header file `*.h`. Arrows indicate logical relationships within bigger modules, however, these relations are high level, they help to understand module interactions and are much more complex in the code. Each bigger square represents a logical group of multiple modules. Also, keep in mind that this grouping is arbitrary as files in C are not grouped, packaged or somehow logically separated.

Figure 3. Logical components of Vim project

In order to explain logical relationship of components, lets start from the main loop, which is being run in `main` module and continue with logical interactions that happen inside of Vim. The `main` module provides necessary initialization and gives control to commands group.

Vim operates on several modes. The most used are `Normal` mode, which executes user commands and `Insert` mode, which changes the contents of the file (which is called `Buffer` in the context of Vim). The basic idea is that Vim waits for the user to type a character and processes it until another character is needed. This is provided in module `Getchar`.

Commands usually change the contents of the `Buffer` (file, as mentioned above). As Vim is quite an old text editor, it used to have quite sophisticated file loading to memory strategies, when operational memory used to be a limiting factor for machines. The logical group called 'Managing files' is responsible for correct file loading to memory, keeping the swap file (in case of the crash) and many other crucial text editor functionality such as character encoding etc.

Vim is a multiplatform program, and I/O on different platforms are managed differently, thus Buffers use `Platform` logical module in order to correctly handle I/O operations on specific platforms.

The most important part of every software application is the functionality it can provide. Vim features such as `Syntax`, `Spell`, `Term` and others are grouped into `Operations` block. It contains modules that are responsible for certain Vim functionalities. This block usually changes `Buffer` and changes how file is presented in 'UI' level.

The `User interface` group obviously is responsible for correct visualization of Vim on the screen. Vim originally is a terminal application, and modules `UI` and `Screen` are responsible for representation and redrawing the terminal. However, separate Vim GUI application was developed over the years and module `GUI` covers it's functionality. Vim commands and operations usually change the screen and trigger the redraw.

Finally, another important part of Vim is 'Core': modules 'Vim' and 'Global' which are mostly included into other modules and contain state information and core code, that is being used in many different modules.

26.5.2   4.2 Common design model

A set of design constraints apply in the Vim project to maximize commonality across element implementations. The common processing and standardization of design is discussed in this paragraph.

- **Common processing:** Common processing standardization of configuration parameter, interfacing and internationalization are applied in each module of the Vim project. The standardization of common processing functions in modular Vim components is accomplished by shared naming scheme, script coding convention and is enforced by strict version control restriction.

  - **Internationalization:** The internationalization of the Vim software command, menu elements and internal help documentation are translated into several languages due to its popularity internationally. Vim's international translation is managed by a collection of translation communities per language and per translation location. The coding convention of the translation is unified within separate modules - translated items are listed in Vim scripting files, and is called by functions (e.g. `menutrans` & `Langstring`) depending on the location of the displayed language and its functionality.

  - **Configuration parameters:** Vim's configuration parameters are highly module-dependent and are stored separately. The editor's mark-up parameters that determine the syntax for different language input and the editor's general configurations are either categorized by the type of programming language or user's preferred interface language. For syntax highlights, indentation type and other programming-related settings, each programming language has a vim script stored as their parameter sets. For message logging, error & bug reporting and other semantic-related, application level settings, parameters are different per spoken language.

  - **Interfacing convention:** Vim's internal interfacing is realized by header file structure in C scripting language that stores a group of functions and declarations exposed as application interface for other files and programs. Each source-code file in the project's `src` folder include `vim.h` and other specific header files to invoke functions internally by header interfacing mechanism.

- **Standardization of design:** Unlike most of the hosted application and project on the distributed version-control system, Vim's core application development took place before Git and Github became the industry standard of code hosting. As a result, the design standardization is divided into a legacy methodology and a modern methodology.

  - **Traditional design standardization:** Vim uses a combination of mailing lists (official and third-party) in the community and development process. Developer's mailing list is resort to Vim contributors and potential developers which offers a platform for discussion of bugs, new features and development issues. In addition, Vim's team relies on a Google group for progress management and issue tracking. Each Google group thread contains a feature / bug description and Github commit summary, where there are links that direct to the relevant pages.

  - **Modern adaptation:** Aside from Google group and mailing list, due to the popularity and level of integration of Github, Vim adapted to the version control platform in 2004 where releasing, issue tracking, code repository, testing and support can be managed in one site. Vim increasingly started to integrate with Github's issue tracker and repository. By taking advantage of issue thread, users can disclose bugs and other technical anomalies to a diversity of developers and contributors on Github. Issues are addressed, on average, within 2 days and commits will be enforced with the support in the corresponding thread and its management team.

– **Coding styles:**

* **Code composition:** Vim's source code is mainly written in C and its configuration parameters are written in Vim script. Each source code file is dedicated to a task or a feature, concatenated with function stacks defined in the header files.

* **Debugging convention:** Debugging of Vim can be achieved in the editor itself through the command `:Termdebug`. Additionally, if the error or bug is time critical or too many dependencies are involved, the channel logging `ch_log` can be added in the source code, and be retrieved by `:call ch_logfile` for debug reference.

* **Documentation:** The documentation of each folder and section of the Github repository is formulated by either Markdown or text format. With `readme.md` as the main documentation for that session and possible `contributing.md` for potential contributions to be submitted by developers.

26.5.3   4.3 Codeline model

In this section code folder structure, building, releasing, testing and deploying procedures are discussed. The chapter is structured based on the "Codeline Models" chapter in Rozanski & Woods.

26.5.3.0.1   Folder structure   Code folder structure of Vim is defined in table 2.

Folder	*Description*
src	C source code, containing mainly loose unorganized files, but also some code organized into folders
runtime	Vim scripts, that are loaded into vim at runtime
ci	Scripts for TravisCI continuous integration
pixmaps	Pictures for Vim GUI application
nsis	Tool that creates one click installation for Windows
READMEdir	It contains readme files for different platforms
src/po	Contains internationalization files (translations etc.)
src/proto	.pro files for generating makefiles for specific platforms
src/testdir	Most tests

Table 2. List of Vim project's folders and their description

26.5.3.0.2   Building   For building code, Vim developers use GNU Make. It ensures that the source code is compiled in correct order. In order to achieve that, developers create `Makefile` where the order and dependencies between multiple source code files are described.

26.5.3.0.3   Testing   In terms of testing, the Vim team is using it's own script called Vimscript. It does not use any standardized testing frameworks. The desired structure for tests is described in a readme file and test creators are responsible for following the structure described therein and correctly adding new tests to the test suite. Some tests use what is called `Old Style Test`, that style is unofficially deprecated but the tests are still present.

26.5.3.0.4   Deploying   Vim is using TravisCI for continuous integration. It runs tests and builds code for Linux and Mac environments after each commit (Windows is handled separately). The tool also generates CodeCov report.

26.6   5. Technical debt

Vim is a very mature project, activity has been mostly steady throughout its 28 years and the core team, lead by Bram, has also been stable throughout. For the last decade or so Vim has been fairly static, not many new features have been implemented, there just aren't many new features left to implement, and so the focus has been more on improving the smaller things. Unfortunately the git history "only" goes back to 2004 but based on comments made by Bram and our analysis of the git history it is clear that this shift in focus from implementing to improving has drastically reduced the technical debt within the code. However a mature project such as Vim suffers from a different kind of technical debt; organisational debt. Through the years the team around Vim has developed a certain way of operating and even though this way of operating may have got Vim to where it is today there are several "organisational smells" that should be considered.

First organisational debt is discussed, second the bus factor, third the automated analysis, fourth the hotspots, fifth the test coverage and as last a conclusion is provided.

26.6.1   5.1 Organisational debt

The backlog for Vim is quite large and non-centralised. There is a large (6k lines) `TODO.txt` file with a list of issues, there is also a backlog in Git and finally there are roughly 800 TODO items in the code itself. Some of these are connected, for example with git issue numbers, but there is no centralized list of issues or overview available. Additionally the organization of communications regarding bugs, feature requests and issues is quite complicated, consisting of a combination of Google Groups, Mailing Lists and Github's inbuilt features. This system works well for current members of the community, who are familiar with the system and know it well, however it is unnecessarily complex. This is typical of many traits in the Vim community, the complexity is well understood by experienced and prominent members but for newer or less experienced members it creates a higher barrier of entry reducing.

26.6.2   5.2 Bus factor

The Bus Factor for Vim is 1. If Bram gets hit by a bus the project will suffer greatly. Bram is the sole Git contributor, by far the most active in commenting on issues and Vim is his creation. Several other members of the community are also important and, combined, would probably have the knowledge needed to continue the project but short-term progress would be severely limited

26.6.3   5.3 Automated analysis

According to our analysis conducted with SonarQube Vim receives an A for technical debt, with an estimated 224 days of work required to eliminate the debt. 224 days might seem like a lot but considering there are a total of 348.000 LoC in the codebase that is not too bad. But in order to get a more focused look we also analysed the git history using CodeScene. CodeScene is a tool that analyses git commits and indicates which files change most frequently and thus should be prioritized with regards to technical debt.

26.6.3.1   Hotspots

Using CodeScene we identified 7 critical hotspots which account for 17% of the total LoCs but were edited in just under 40% of the commits analysed. Based on this we decided to further analyse the technical debt and code smells found in those 7 classes. As can be seen in table 3 technical debt for each of these classes seems to be consistent with the size of each class compared to the rest of the project. However we found a total of 174 functions across these 7 classes whose cognitive complexity was above the 25 recommended by SonarQube, one of which had a cognitive complexity of 1762 (and an estimated technical debt of more than 3 days)! In table 3 you can see that the complexity of these functions accounts for a majority of the technical debt in the hotspot classes and that is not considering other technical debt within these functions. An informal analysis shows that these complex functions are also the ones that are changed most frequently so we highly recommend that these functions be simplified where possible. Interested readers can access the SonarQube analysis here

Class	Commits	LoC	Technical Debt	Reducing Complexity
eval.c	421 (8.7%)	8348 (2.4%)	5d 1h (2.3%)	3d 5h
terminal.c	296 (6.1%)	4975 (1.4%)	2d 1h (0.9%)	1d 1h
channel.c	287 (5.9%)	4829 (1.4%)	2d 2h (0.9%)	1d 4h
evalfunc.c	276 (5.7%)	11846 (3.4%)	5d 2h (2.3%)	2d 4h
option.c	226 (4.7%)	10997 (3.2%)	7d 1h (3.2%)	5d 5h
ex_docmd.c	214 (4.4%)	9701 (2.8%)	7d 4h (3.2%)	4d 7h
screen.c	203 (4.2%)	8389 (2.4%)	9d 4h (4.1%)	7d 7h
Total	1923 (39.7%)	59085 (17.0%)	37d 15h (16.8%)	24d 9h

Table 3. List of results from CodeScene and SonarQube

26.6.3.2   Test Coverage

Vim has been using Coveralls to monitor test coverage since 2015 and during that time coverage has gone up from a steady 58% to just under 80% today. The Vim community is aware of the need for improved coverage and therefore any PR that reduces test coverage gets flagged, however this does not necessarily mean that the PR will be rejected. Aside from the level of coverage there does not seem to be much of a structure to the test suite. Almost all tests are contained in a directory called /src/testdir with only a few tests contained in subdirectories and while newer tests have names indicative of what is being tested older tests are simply identified with a number. All of this combined leads to a fairly disorganized test suite

which will continue to grow and become even more disorganized if nothing is done. If the Vim community is sincere in their goal to improve the test suite this problem needs to be addressed.

26.6.4   5.4 Conclusion

Ultimately Technical Debt is a measurement of the cost of introducing new features or improving old ones. It is fair to say that in the case of Vim this cost is very low. We estimate that over the last year there have been on average around 20 new releases every week. An informal analysis shows that the time from when a bug is reported in Git until it a fix has been released is usually less than a week. For the long-term future of Vim there are several organisational issues that need to be fixed, these range from simply inconveniencing new community members to possibly endangering the future of Vim. Vim has gotten to the point where it will soon be time to consider what happens when core members retire due to old age and new members need to take their place, before that time comes, Vim will have to have made some changes.

26.7   6. Evolution perspective

The evolution perspective describes all of the possible types of changes that a system may experience during its lifetime. First a brief history of the development of Vim is provided, second the evolution of Vim is discussed in which especially the lack of documentation, the context view, development view, technical debt and competitors are discussed.

26.7.1   6.1 History of Vim

Bram Moolenaar released the first version of Vim in 1991 and continued developing Vim during the years. An overview of the history of Vim can be found in figure 4 [8].

Figure 4. History of Vim

26.7.2   6.2 Evolution of Vim

Vim is currently in a stable deployed state with a stable core. Vim is highly flexible and many developers personalize Vim by adding add-ons.
The Developer Survey Results of Stack Overflow of 2018 show that Vim has a market share of around 25% [9]. Since so many developers are used to the way Vim works and think Vim is sufficient for now, they probably will not change to another text editor any time soon. But what will happen to Vim in the coming years when the technical environment changes? We don't know what technical developments will come or when they will come, but we can assess how Vim will do in future technical environments.

26.7.2.1   Lack of documentation

One of the main issues for the evolution of Vim will be the lack of documentation. This prevents developers who do not know a lot about Vim's code to alter the code and become invested in the project. This small
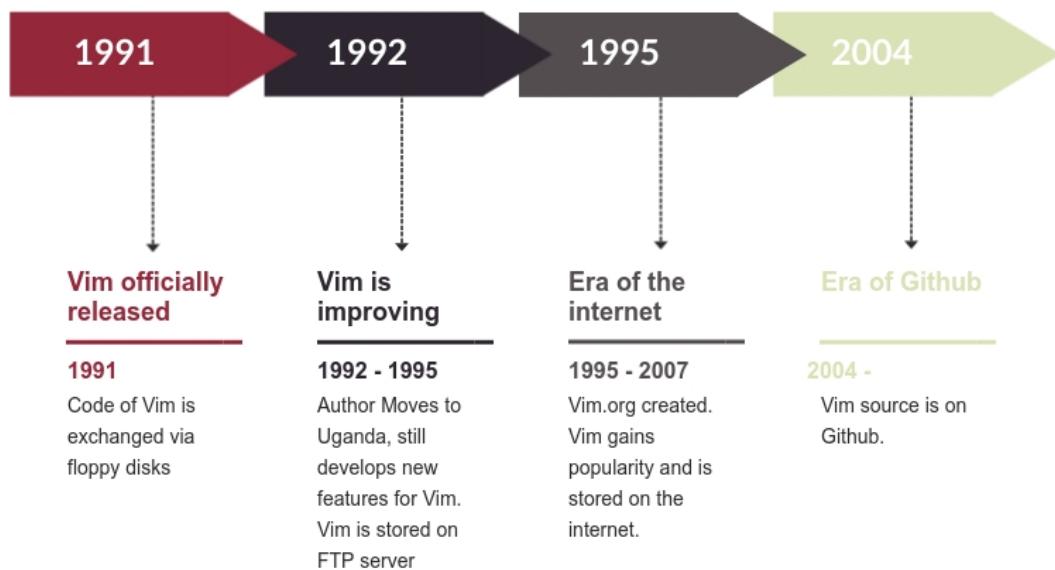
Figure 26.3: History of Vim

preservation of knowledge might become difficult when people move to other projects, memories fade and the available technical environment changes.

26.7.2.2 Context view

The context view will change in the way that the core developers and other developers might become different people. Since Bram is currently the only integrator in the project, if he becomes less active it might happen, that developers will make their own versions of Vim and thus multiple branches of Vim will exist. Another possibility is that the developers will continue working on this version of Vim, which may or may not be a copy of the Vim GitHub repository.

26.7.2.3 Development view

The most probable code needing to be altered in new technical environments is the code related to the operating systems Vim runs on. With new technologies new operating systems will be developed and thus Vim will have to adjust.

26.7.2.4 Technical debt

Because of the lack of documentation, new developers have a high probability of increasing technical debt. If Vim remains only the stable core, the technical debt will not change much and will probably continue

being reduced. However, this is reliant on the core continuing its effort, if their effort is diminished technical debt might increase or development slow down.

26.7.2.5   Competitors

Because of multiple reasons Vim might lose market share in the future. The first reason is because Vim is mostly written in C and this programming language is becoming less used and learned, because of this less people will be able to understand and alter Vim's code. The second reason is because of Vim's interface, the current interface offers a lot of possibilities that a developer might want and need, but it looks more old-fashioned compared to some competitors. The third reason is because of the functions Vim offers, some developers prefer other functions than the ones Vim offers and thus decide to use a competitor instead of Vim.

26.8   7. Conclusion

After close to 30 years of existence, Vim is a mature project, set in its ways on knowing exactly what it wants to be, its community is as active as ever with a considerable crowd participating in discussions every day. The user group is stable and under the leadership of its creator, Bram Moolenaar, so is the development community. Despite being created in a world where memory was a serious constraint, the internet didn't exist and most of current programming languages and tools such as Github or Java were a distant future, Vim has managed to adapt to the incredible changes in IT over the last three decades and is as relevant as ever. Yet certain core elements have not been adapted to more modern views on software architecture, especially parts of file organization, workflow and communication patterns. Today the main focus of Vim is to continue to make sure that Vim will be available in as many platforms as possible and maintenance of current functionality, the team is semi-actively working towards improving things such as test coverage and documentation. There is no reason to expect anything other than a bright future for Vim.

26.9   References

[1] Rozanski, N., & Woods, E. (2011). *Software systems architecture: working with stakeholders using viewpoints and perspectives*. Addison-Wesley.
[2] Gousios, G. (2014). How do project owners use pull requests on Github? Retrieved February 15, 2019, from http://www.gousios.gr/blog/How-do-project-owners-use-pull-requests-on-Github.html
[3] List of prominent contributors of Vim (2019). Vim's documentation. http://vimdoc.sourceforge.net/htmldoc/intro.html
[4] Specification of Valgrind check on Vim's codebase (2019). Vim's documentation. http://vimdoc.sourceforge.net/htmldoc/debug.htm
[5] List of prominent Vim's sponsor (2019). Vim's documentation. https://www.vim.org/sponsor/hall_of_honour.php
[6] Destination of Vim's support and sponsorships. Vim's documentation. http://vimdoc.sourceforge.net/htmldoc/uganda.html
[7] Moolenaar, B. (2019). Vim's Github repository. https://github.com/vim/vim
[8] Moolenaar, B. (2016). Vim 25 presentation by Bram Moolenaar on 2016 November 2 [Video]. Retrieved from https://www.youtube.com/watch?v=ayc_qpB-93o&feature=youtu.be
[9] Stack Overflow (2018). StackOverflow's developer survey results 2018. https://insights.stackoverflow.com/survey/2018/#technolog most-loved-dreaded-and-wanted-platforms

26.10 Appendix A: Analysis of pull requests

This appendix contains a complete overview of the PRs analyzed for the merge decision strategy, including the information obtained from each of them. The PRs are grouped into accepted or rejected PRs.

26.10.0.1 Accepted pull requests

Pull request 1:
PR number: 4011
PR name: Large files were needlessly truncated to 2147483647 lines.
Proposer: dpelle
Accepted or rejected: accepted
Problem: A very long file is truncated at 2^31 lines.
Solution: Use LONG_MAX for MAXLNUM.
Comment: There has been some discussion on the PR and multiple people agree on the PR.

Pull request 2:
PR number: 4004
PR name: Fix incorrect 'mouseshape' documentation about the defaults
Proposer: ychin
Accepted or rejected: accepted
Comment: Documentation update is committed without comments

Pull request 3:
PR number: 3974
PR name: Scroll over ten times speed in vim.exe
Proposer: ntak
Accepted or rejected: accepted
Problem: When using VTP scroll region isn't used properly.
Solution: Make better use of the scroll region.
Comment: PR was proposed, Bram asked if others wanted to say whether this was a good idea, one person said it was, Bram made the commit.

Pull request 4:
PR number: 3969
PR name: Fix a broken verblose messages
Proposer: mattn
Accepted or rejected: accepted
Problem: Message written during startup is truncated.
Solution: Restore message after truncating.
Comment: mattn found a problem, later offered a patch, k-takata offered another patch, they discussed both patches and k-takata mentions (@) Bram who then committed one of the patches

Pull request 5:
PR number: 3968
PR name: Allow the window to shrink in vim.exe
Proposer: ntak
Accepted or rejected: accepted

Problem:MS-Windows console resizing not handled properly.

Solution: Handle resizing the console better.

Comment: itouen made an issue with number 3611 in November 2018, at that time no comments or commits were made. The same problem was stated in this PR. ntak states a problem and k-takata and ntak discuss a possible solution, they agree and the PR is committed by Bram.

Pull request 6:

PR number: 3967

PR name: Add error E982 when ConPTY is not available.

Proposer: h-east

Accepted or rejected: accepted

Problem: No error when requesting ConPTY but it's not available.

Solution: Add an error message.

Comment: h-east made a PR and this was committed by Bram without any further information

Pull request 7:

PR number: 3954

PR name: Re-enable find_module: it was deprecated, not removed

Proposer: joelfrederico

Accepted or rejected: accepted

Problem: With Python 3.7 "find_module" is not made available.

Solution: Also add "find_module" with Python 3.7.

Comment: joelfrederico mentioned a problem, Bram made a fix and committed this. joelfrederico was very satisfied.

Pull request 8:

PR number: 3855

PR name: Proposal: Add some system names to feature-list of has()

Proposer: ichizok

Accepted or rejected: First rejected, later accepted

Problem: Not easy to recognize the system Vim runs on.

Solution: Add more items to the features list.

Comment: ichizok made a PR, multiple people commentented, which lead to ichizok altering the PR, commenting and altering happened multiple times and finally Bram committed the PR.

26.10.0.2   Rejected pull requests:

Pull request 9:

PR number: 3929

PR name: Fix: can add the new value to "a:" dict

Proposer: ichizok

Accepted or rejected: rejected

Comment: ichizok made a PR, multiple people, including Bram, had a discussion on whether it was a good idea to commit it. They decided it was not a good idea and so it was not committed.

PUll request 10:

PR number: 3946

PR name: Change 'termwintype' default to "winpty"

Proposer: h-east
Accepted or rejected: rejected
Comment: h-east made a PR. Someone did not agree with this solution to the problem and proposed another solution. h-east agreed the other solution is better and closed his own PR.

Pull request 11:
PR number: 3919
PR name: define RESTRICT
Proposer: mattn
Accepted or rejected: rejected
Comment: mattn made a PR, mgedmin commented that the PR was pointless and gave a reason. mattn closed the PR.

Pull request 12:
PR number: 3867
PR name: Do away with create_cmdidxs.vim and automatically update ex_cmdidxs.h
Proposer: rbtnn
Accepted or rejected: rejected
Comment: rbtnn made a PR, Bram commented that the situation was very rare and the PR would not lead to a lot more convenience. rbtnn agreed and closed the PR.

Pull request 13:
PR number: 3816
PR name: Add :vsb[uffer]
Proposer: bobrippling
Accepted or rejected: rejected
Comment: bobrippling made a PR, someone commented that the problem he had was solved by a plugin and thus bobrippling closed the PR.
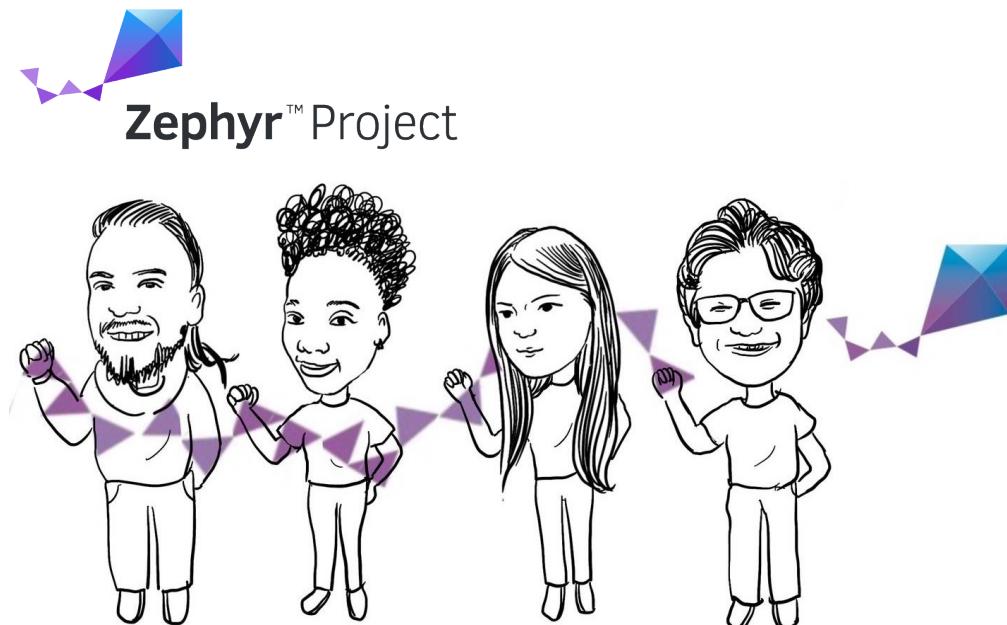
Chapter 27

Zephyr

By Jure Vidmar, Eghonghon Eigbe, Oxana Oosterlee, Suryansh Sharma

Delft University of Technology, 2019

27.1   Table of contents

27.2   1. Introduction

The Zephyr Project is an open source collaboration project hosted by the Linux Foundation. It is a small, scalable, real-time operating system (RTOS) built for embedded platforms with diverse hardware architectures. It is aimed at Internet of Things (IoT) devices, which often have stringent resource constraints. Therefore, the Zephyr Project's goal is to design a modular operating system that has a small footprint and low power consumption. Furthermore, it places a high importance on network security to protect devices connected to the internet from potential harm. Because of its modularity, the software can be developed easily for different platforms with different specifications. It contains a lot of tools such as sensor and device drivers and a networking stack to make it easy for developers to fit the software to their specific application. It is an open-source project that has a large network of contributors, including a number of industry partners, and the community can evolve the project to support new hardware, developer tools and drivers.

27.3   2. Stakeholder Analysis

Different people have different interests in the Zephyr Project. To identify these different stakeholders, an analysis was done using the stakeholder classes as defined by Rozanski and Woods [1] as a guideline.

Type	Description
Acquirers	The Zephyr Project is acquired by the Linux Foundation, which provides a legal and administrative framework for the project. Funding comes from companies that can become a 'member' by paying an annual fee.
Assessors	The project's governing board sets project goals, makes marketing and legal decisions and oversees the budget.
Developers	There is a large community of regular contributors. The core developers are mainly employees of the member companies. They manage the system and make major contributions to it. Key developers identified are presented in Table 1 below.
Communicators	A community sub-group is responsible for the management and coordination of community activities and communication. There are weekly group meetings for the different working groups and committees that are open to the public, except for the security working group. All user and developer documentation is provided through the Zephyr Docs page.
Maintainers	The technical steering committee sets the direction for the project, sets up new projects, coordinates releases, enforces development processes and moderates other working groups within the project. It is chaired by Anas Nashif (Intel). A system of codeowners exists, in which the core developers of the project are responsible for their parts of the system and review every code change that affects these parts. One of their main challenges is maintaining code quality, which is necessary in order to provide secure and functionally safe, certifiable RTOS.
Suppliers	Manufacturers of hardware platforms provide an environment on which the Zephyr RTOS can be deployed. The Zephyr kernel itself has been derived from Wind River's commercial VxWorks Microkernel Profile made for VxWorks. The Linux Foundation also currently provides some servers and infrastructure to the project.
Support Staff	An administrative advocacy sub-group focuses on managing public relations, the website as well as training and materials management and other outreach activities. Technical support comes directly from the project's active contributors, who can be contacted through different communication channels (e.g., Slack).
Testers	A Testing Working Group oversees the testing procedure and tools. These tools include the testing framework available (Ztest) and the Continuous Integration.
Users	Zephyr is built for embedded programmers building real-time and connected (IoT) systems on resource constrained devices. These can be both individual users as well as commercial organizations that can use the software under the Apache v2.0 License.
Competitors	Direct competition comes from other IoT targeted RTOSes, such as Riot OS and ARM Mbed OS. To a lesser extent it competes with projects as freeRTOS and commercial OSes such as μC/OS and VxWorks, which provide IoT support as part of a larger system.
External Regulators	The Zephyr Project is subject to export compliance set by the US Export Administration Regulations and other U.S. and foreign laws and may not be exported, re-exported or transferred to a list of countries set by the US government.

Type	Description
Security Evaluators	One of the distinguishing features of Zephyr is the security it promises. Any changes that appear to have an impact to the overall security of the system need to be reviewed by a security expert from the project's security working group. This working group consists of employees of the member companies.

Figure 1 shows the most important stakeholders in a power-interest graph. The project's organisational bodies clearly have the largest power within the project, where the member companies play a big role in providing the manpower. These are mainly internal stakeholders. External stakeholders have less power, and their interest in the project is mainly based on their usage of Zephyr.
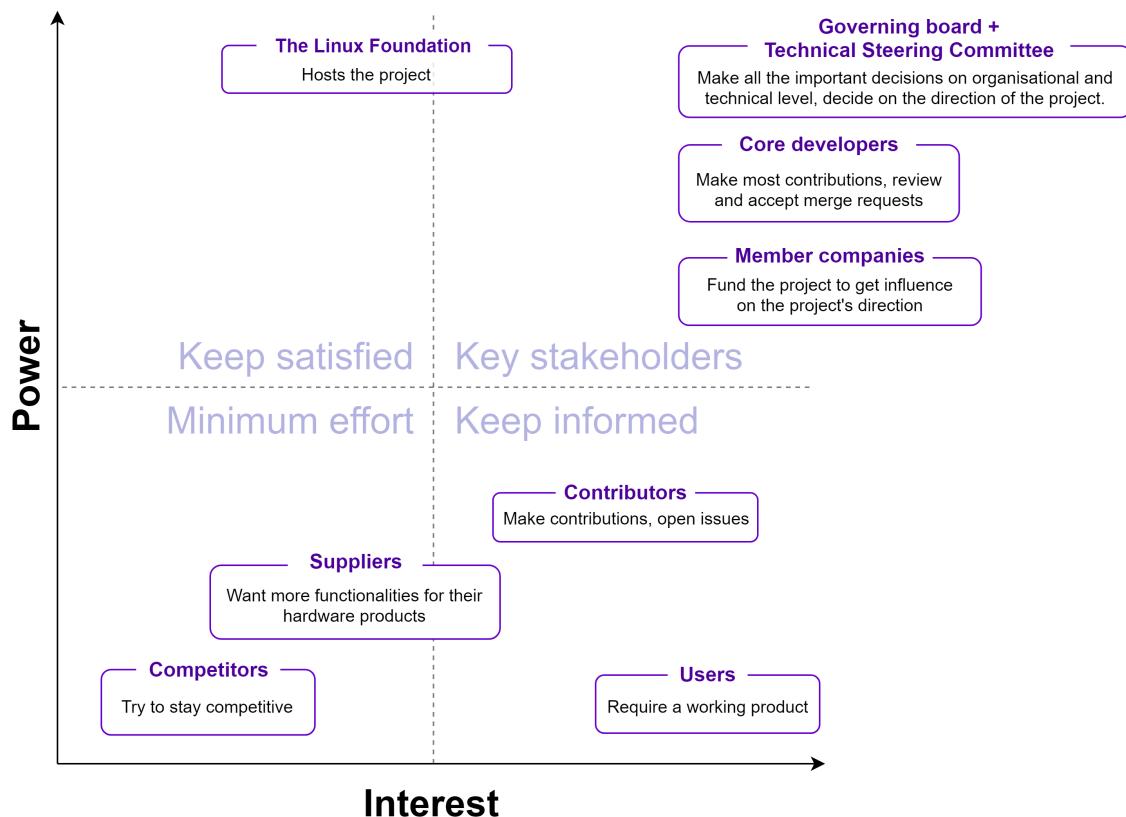

Figure 1 - Power-interest graph

Table 1 - Key Contributors to the Zephyr Project

Github ID	Affiliation	Contribution (out of 8025 closed pull requests)	Role
@nashif	Intel Employee	599	Reviewer, Integrator, Developer

Github ID	Affiliation	Contribution (out of 8025 closed pull requests)	Role
@galak	Intel Employee	559	Reviewer, Integrator, Developer
@carlescufi	Nordic Semiconductors Employee	225	Reviewer, Integrator, Developer
@mike-scott	Independent	122	Reviewer, Developer
@SebastianBoe	Nordic Semiconductors Employee	285	Reviewer, Developer

27.4   3. Context view

From the stakeholders discussed in the previous section, it is clear that a lot of internal and external entities are involved in the Zephyr Project. To provide context on who/what these entities are and how they are connected, the context view in Figure 2 presents an overview of the most important entities.

The project is well-organised by its **host** the Linux Foundation. Its **governance** consists of two main bodies: a governing board that makes project management decisions and a technical steering committee (TSC) that manages the development. The TSC chair sits in both boards and provides a link between the two. Specialized committees such as the security working group fall under the responsibility of the governance. Members of the boards are provided by the **project member** companies, who pay a yearly contribution. The membership is awarded in tiers which are based on the contribution amount. A higher tier means more influence in the project's governance.

Zephyr is **developed** by a community that consists of 400+ contributors. Most of the main contributors are employed by member companies. For **communication** Github and the mailing lists are used, and to a lesser extent the Slack and the IRC chat channel.

As **development tools**, Zephyr relies mainly on GitHub, using Shippable for Continuous Integration. C is the predominant **language**. **Documentation** can be found on Zephyr's docs website, and is generated using both manual written documentation files that are kept in the source directory, as well as using Doxygen for automatic generation.

Zephyr runs on multiple different **architectures**, and is ported to a continuously increasing list of 150+ boards. **Boards** that are supported are for example from manufactures as STM (Nucleo boards), Arduino, Espressif, NXP and Nordic Semiconductor. **Competitors** that also provide support for (a subset of) these platforms are for example Riot OS and ARM Mbed OS. Users can use any of the supported boards in their project. Commercial users can use Zephyr under the Apache v2.0 **license**. Current examples of users include CommSolis, an ultra-low power solutions company for the Internet of Things (IoT) and the Philae and the accompanying Rosetta Orbiter which landed on Comet Churyumov–Gerasimenko.
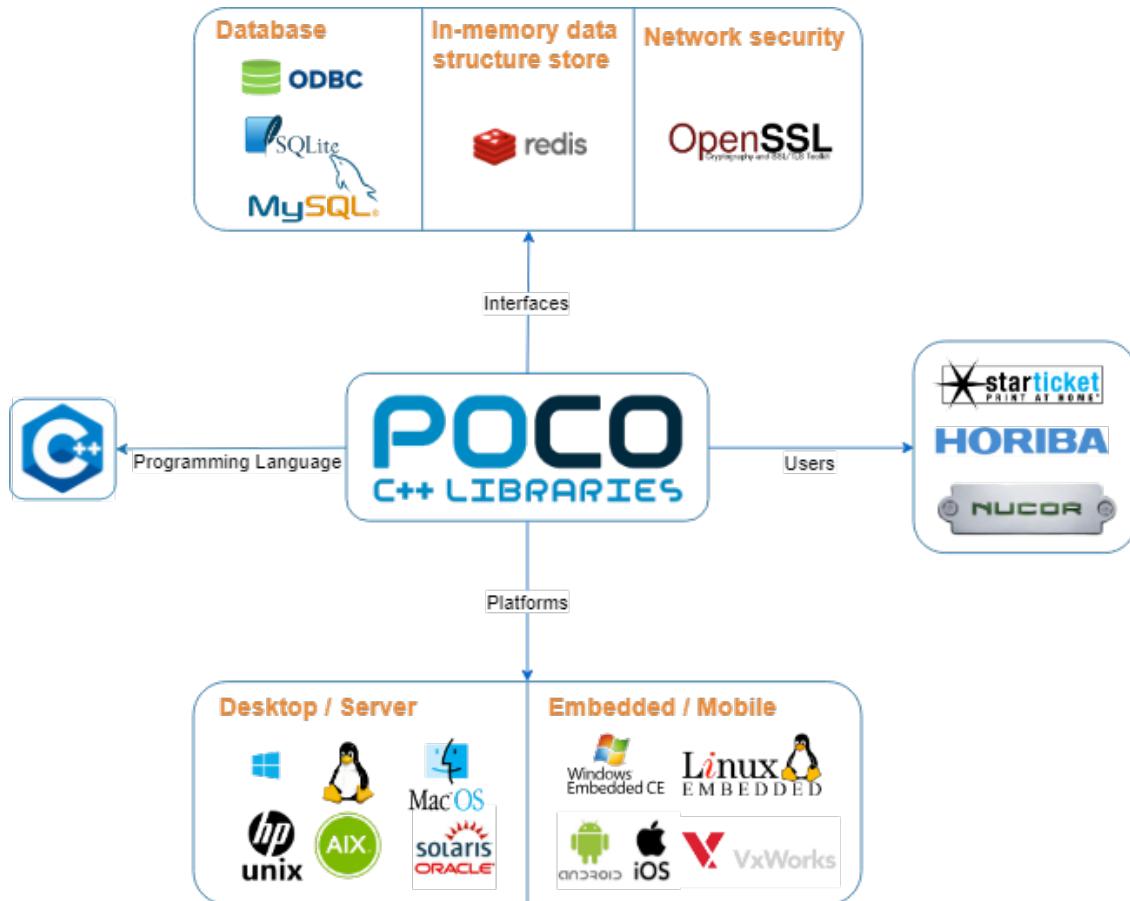
Figure 2 - Context view

## 27.5   4. Decision Making Process

This section discusses how pull requests are handled within the Zephyr Project. The analysis was done by first clustering issues into different classes, followed by an analysis on a number of pull requests, which are listed in the Appendix.

Figure 3 shows the groups of issues, the concerns they address and the stakeholders who influence these issues the most.
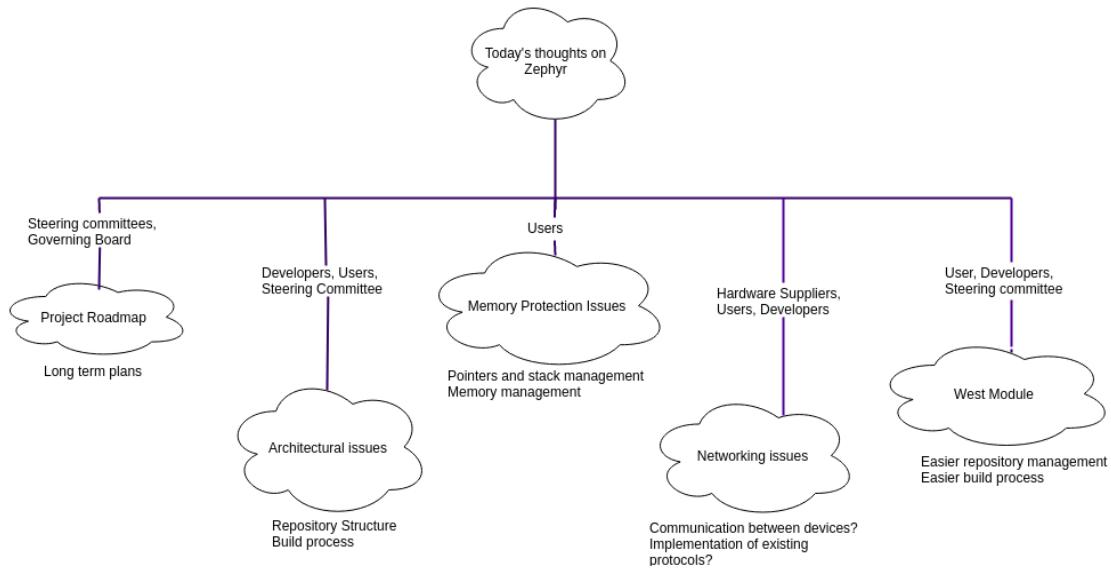
Figure 3 - Issue Clusters

27.5.1 4.1 Decision Making Process

When a pull request is made, an automatic tool *Codecov* analyzes it for code coverage and comments with the percentage of the code the requested pull attempts to change. The Continuous Integration also performs different analysis and sanity checks after which it is reviewed by the designated reviewers. Following their approval or otherwise, a decision is reached by the integrators whether to merge or decline the request. In the process of making this decision there is an option to request changes from the developer which can change the decision of a reviewer. There is also a back and forth that gives the developer a chance to justify or clarify aspects of the request before a decision is made. To merge a request, all the assigned reviewers must approve the changes made. This process of approval was noted to be influenced by the following factors: 1. The relevance of the pull request to the project. Not every pull request was a direct response to a published issue and in cases where they were due to developer initiative, developers had to justify why this feature or change was required and sometimes how it would affect other components of the project.

2. The extent to which the change alters the project. This is indicated by the *Codecov* analysis discussed above but is also sometimes evaluated by developer intuition and experience with the project. A typical instance is a pull request that attempts to change core kernel functionality like the operating system scheduler. Such requests are put under more scrutiny than others.

3. Code quality. Usually in terms of following coding style and common practices within the project.

4. Precedence issues. There are cases in which two pull requests overlap in the issues that they solve. In such cases, a pull request is selected over the other based on the factors discussed above with one being merged and the other declined.

The flow is shown in Figure 4.
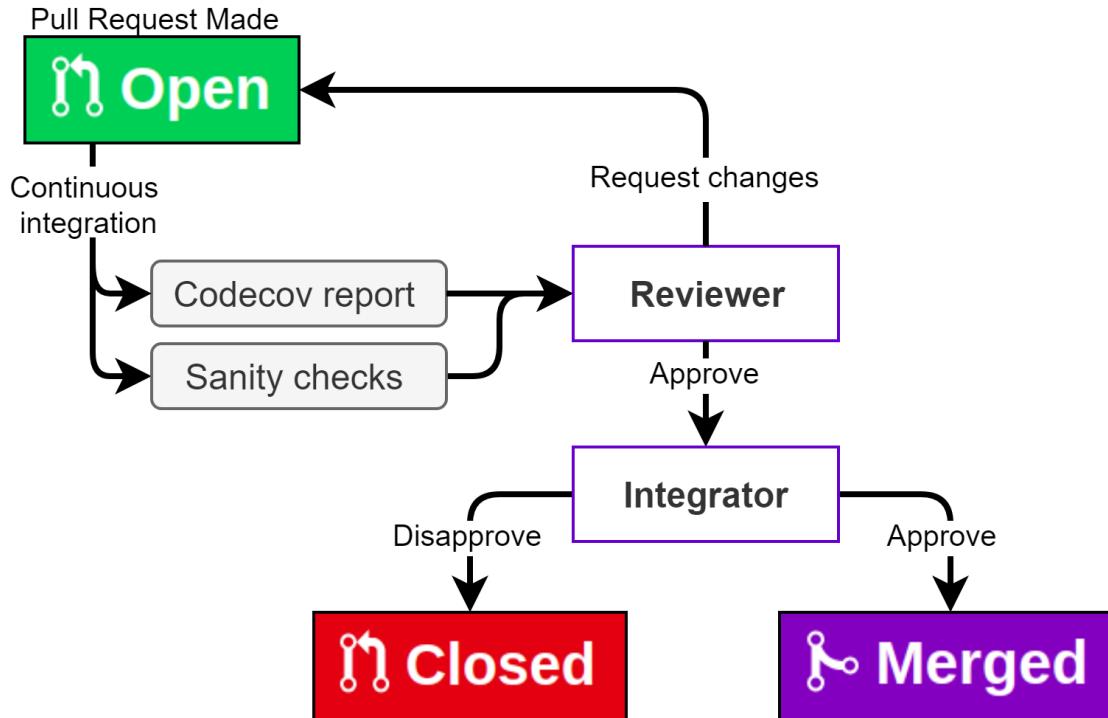
Figure 4 - Pull request flow

27.6 5. Development View

The Development View provides an overview for the developer on how the codebase is organized, what standard design practices are in place, and how the codebase is managed for different build configurations.

27.6.1 5.1 Codeline Organization

The Zephyr Project aims to provide all required building blocks needed to deploy complex IoT applications. It organizes itself across multiple Git repositories, separating the RTOS code from external libraries and development tools. This improves modularity, avoids conflicts in licensing and certification, and enables out-of tree development. Dedicated multi-purpose command line tool called 'west' was developed, for easier managment of repositories and development. Figure 5 shows a typical (west) installation directory structure.

```
zephyrproject/
    .west/
        config
        west/    This contains the west source code
    zephyr/      Contains the source code for zephyr
        west.yml   Lists the external repositories managed by west
    external-project-a  Externally maintained projects (vendor HALs, crypto libraries, etc)
    . . .
```

Figure 5 - Zephyr project tree structure

The layout of the base directory which hosts Zephyr's own source code, its kernel configuration (Kconfig) options and its build definitions is shown in Figure 6.

Zephyr

    west.yml This lists the external repositories managed by the west command-line tool.

    CMakeLists.txt The top-level file for the CMake build system

    Kconfig The top-level Kconfig file(other Kconfig files exist at the architecture and board levels)

    soc SoC related code and configuration files.

        arm Arm specific SoC configuration

        x86_64 x86 specific kernel source files, kernel include files for private APIs, Kconfig and CMake files
        . . .

    kernel Zephyr kernel code (architecture independent portions)

    misc Miscellaneous code(currently only contains automatically generated code)

    doc Zephyr technical documentation source files

    dts Device tree source files used to describe non-discoverable hardware details

    arch Architecture-specific kernel and system-on-chip (SoC) code

        arm Arm specific kernel source files, kernel include files for private APIs, Kconfig and CMake files

        x86 x86 specific kernel source files, kernel include files for private APIs, Kconfig and CMake files
        . . .

    boards Board related code and configuration files.

        arm Contains directories for specific arm boards

            stm32_min_dev Contains Kconfig and other config files specific to this board

            96b_carbon Contains Kconfig and other config files specific to this board
            . . .

        x86 Contains directories for specific x86 boards
        . . .

    drivers Device driver code.

    lib Library code, including the minimal standard C library.

    samplesSample applications that demonstrate the use of Zephyr features

    cmake Additional build scripts

    tests Test code and benchmarks for Zephyr features. This also contains the sanity check.

    scripts Various programs and other files used to build and test Zephyr applications e.g. *Gitlint* and *checkpath*

    subsys Code for Zephyr subsystems.

        bluetooth Bluetooth host and controller

        usb USB device stack code.
    . . .

    include Include files for all public APIs used by Zephyr

    ext Contains code from external projects integrated into Zephyr

Figure 6 - Zephyr source directory tree structure

A Zephyr application in its simplest form has the following contents:

app/
    CMakeLists.txt Tells the build system where to find other application files and links the application directory with Zephyr's CMake build system
    prj.conf Kernel configuration file that specifies application-specific values for one or more kernel configuration options.
        These application settings are merged with board-specific settings to produce a kernel configuration.
    src
   **main.c** One or more application-specific files written in C or assembly language

Figure 7 - Zephyr app directory tree structure

27.6.2 5.2 Module Organization

On an architectural level, the source code can be divided into modules - a collection of code serving a specific purpose. The modules can be divided into three layers based on how close a module is to the hardware:

- **Kernel Layer**: The backbone of the OS. Includes modules for managing low-level processes directly related to the hardware and scheduling of tasks.
- **OS Services Layer**: Provides access to all common OS functionalities. The modules in this layer can be seen as building blocks for designing applications.
- **Application Services Layer**: Using functionalities provided by the OS services, users can make applications to implement specific functionalities according to their own project requirements. An application can be for example reading a sensor and sending the data over Wi-Fi.

In each layer, a module can use modules from the same and each lower-level layer. Figure 8 shows an overview of the three layers and a subset of the modules in each layer.
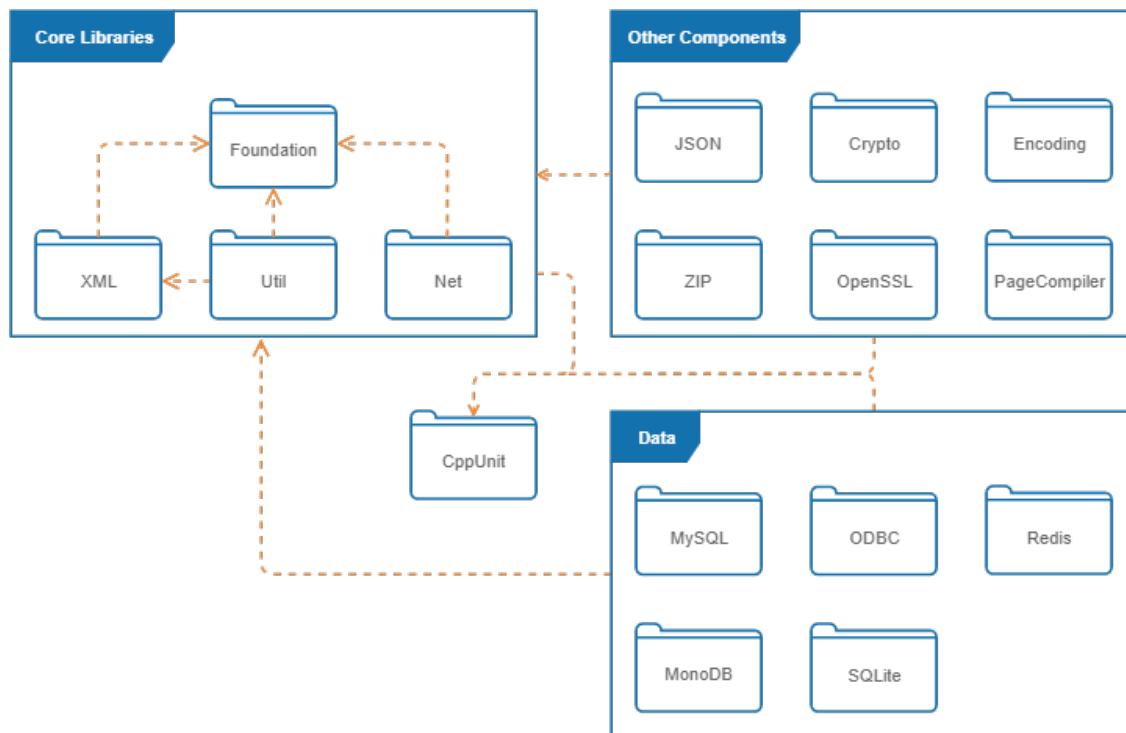

Figure 8 - Organization of Modules within the Zephyr source code. Kconfig files are used for managing variability of different architectures and boards. - note: for clarity only a subset of the modules is listed.

27.6.3 5.3 Build Configuration

The Zephyr build system is application-centric, as applications configure and initiate the build process. The kernel and application are compiled into a single binary using Zephyr SDK toolchains. Developers can also

use their own toolchains for compilation, however. CMake build system is used, allowing two different formats:

- Unix Makefiles
- Ninja

One of the advantages of Zephyr's distributed configuration system is separation of application-specific kernel settings from board-specific settings. At build-time, configuration (Kconfig) files from all layers are combined into a single kernel configuration.

27.6.4   5.4 Variability Management

Variabiity is managed in the project by maintaining board specific code for each platform the operating system supports. Aplication developers make use of configuration files to specify what versions of code is compiled for a project. These files are shown in Figure 7 above under *prj.conf*.

27.6.5   5.5 Instrumentation

For aiding developers in the process of debugging and testing, the Zephyr Project provides the following standardized tools:

- **Debugging** - Debugging can be done using the GNU Debugger in QEMU, which is an open-source machine emulator and virtualization software. When building an application, the build system automatically generates an ELF file that can be used for debugging purposes.

- **Logging** - Logging can be done using the Logger API. It provides logging functions with four different severity levels (error, warning, info and debug) that can be used for debugging RTOS modules. It is configurable via Kconfig files.

- **Testing** - The Zephyr Test Framework (Ztest) provides two methods of testing to the developer: unit testing for testing specific modules and sanitycheck for testing the full software stack. The latter runs a number of different tests (which can be extended by developers) for different architectures using the QEMU emulator. On a daily basis, tests are automatically built and run on the whole project codebase to verify its operation as part of their CI process (using shippable).

`run` `pending`

27.6.6   5.6 Standardization of Design

The success of Zephyr's architecture can be attributed to it's standardization of development. Following the philosophy of "*standing on the shoulders of giants*", the project adopted development patterns and practices that were proven to work well in the industry:

- **Release model** - The project's release model follows the Linux kernel's model with several adaptations. Releases are time-based (rather than feature-based), following roughly three-month cycles. Each release period consists of a *merge window*, where new features and changes can be added, and a

stabilization phase where only bug fixes and documentation improvements are allowed to be merged into the master branch.

- **Development guidelines** - These are also inspired from industry's senior players. Coding style follows the Linux Kernel's style, with minor adaptations. A few examples of code style requirements are: "every `if-else` statement needs braces", "C-99 style of single line comments (`// comment`) is not allowed" and "use spaces instead of tabs for aligning comments after declarations".

- **Naming conventions** - unlike desktop operating systems, where applications are written in user-space, Zephyr's applications are written in kernel-space. To prevent variable naming conflicts in this shared namespace, kernel developers defined several naming conventions, for example, kernel-specific functions are prefixed with `k_`, operations called by Interrupt Service Routines should be prefixed with `isr_` etc. Individual RTOS subsystems can define their own conventions but need to have a subsystem prefix such as `bt_` for Bluetooth stack or `net_` for IP stack, etc.

27.6.7   5.7 Git workflow

The Git workflow is well defined and documented, enabling developers to communicate about the code quality and bugs. This is crucial for resolving and avoiding technical debt, discussed in the next section. The majority of the communication about the source code is done through GitHub issues, which are used for reporting bugs as well as proposing features or enhancements. In the latter case, the proposal is expected to include elaborate description of the problem and the proposed implementation. Before the feature is agreed upon, developers discuss if the proposal fits into the current architecture and give advice on its implementation and maintenance. Large changes are discussed within the Technical Steering Committee meetings.

Commit messages should include relevant information and the developer's signature to DCO (Developer Certification of Origin). Each contribution (pull request) is automatically checked by the Continuous Integration service. Furthermore, each pull request needs to be reviewed by at least two reviewers before being merged into the master branch.

This open review system works very well for the Zephyr Project, ensuring that no big changes are done to the source code without having been properly thought through by multiple (core) developers within the project. Furthermore, having the discussion open to view by anyone adds to the traceability of the code. In case of a large contribution or issue, the discussion can continue on other communication channels such as the mailing list and weekly developer calls.

27.7   6. Technical Debt

In this section the performance of the Zephyr Project in regards of technical debt is analyzed. First, we take a look at what measures are taken now to avoid technical debt. Then, using tools we analyze what the state of the codebase actually is and how the project evolved over time.

27.7.1 6.1 Communication of Technical Debt

As already mentioned in Git Workflow, Zephyr requires all issues to be tagged properly. This combined with their extensive communication process is essential to avoid and resolve technical debt. For technical debt related issues, the following tags are used explicitly:
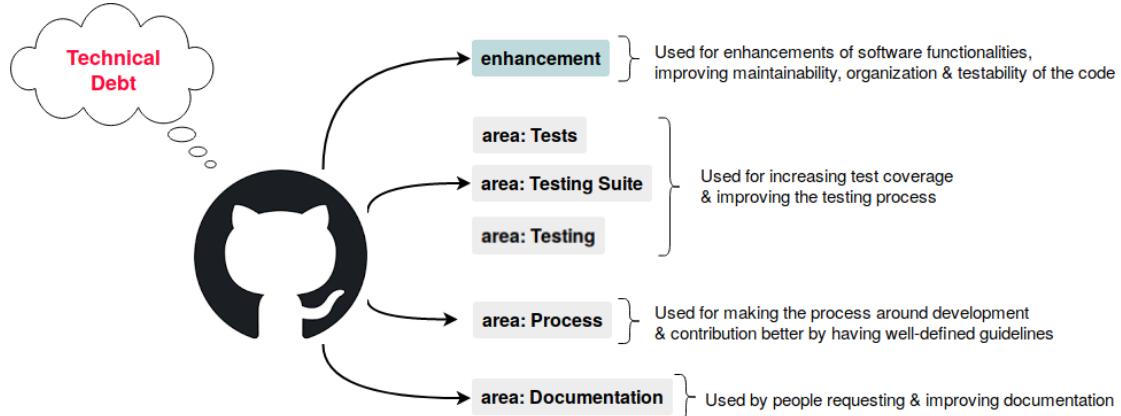

Figure 9 - Tags used for technical debt-related issues

Issues about improving the quality of the code or refactoring do not have a specific tag and can be found under `enhancement`. There exists a tag for this purpose, `area::Code Style`, but this tag is not used often. We find that Zephyr developers fix code issues as part of the tightly controlled development process rather than as an afterthought.

27.7.2 6.2 Automatic Tools for Avoiding Technical Debt

The Zephyr Project uses some automated tools that are run at every pull request as part of the Continuous Integration. These tools contribute to reducing the risk of (future) technical debt. If a pull request does not pass the tests all automated tools run, the pull request will not be merged.

Avoiding technical debt related to the source code:

- *Codecov* - Generates a report on how test coverage of the full project changes with the new pull request added. A high coverage percentage means a lower testing debt.

- *Sanitycheck* - Checks if all tests still pass on a range of different board builds.

- *Checkpatch* - Checks the pull request whether the code complies with the coding style guidelines and naming conventions.

Avoiding technical debt related to documentation and traceability:

- *Gitlint* - Checks if a commit message is written according to the commit guidelines. Good commit messages improve traceability.

- *Documentation style check* - Checks if added documentation adheres the documentation style guidelines.

They follow some of the best CI practices as can be seen from their gold badge and this process goes a long way in avoiding technical debt.

27.7.3   6.3 Automated Analysis

This section presents the results of running the Zephyr repository through Sonarqube, which is an automated analysis tool for code quality. Results are presented for code reliability, maintainability, duplication, coverage, complexity and size as these are deemed the most relevant to the project. These metrics are specifically relevant because of the hard real time use cases of Zephyr (and other RTOSes) which require predictable and easy to analyze code.

Figure 10 - Sonarqube analysis overview

27.7.3.1   6.3.1 Duplication

The analysis gave a passing grade but raised 3400 code smells. Upon looking into these, it was clear that they were mostly from duplicated blocks of code. Putting this in context, we are reluctant to brand this a debt because it has a lot to do with how Zephyr manages variability. The largest contributors to the count are files from *ext*, *include* and *build* shown in Figure 6. It is an operating system that supports a large variety of boards and build files are maintained for different boards with multiple boards using the same dependencies. Upon building an application based on Zephyr, only board-related code is ported to the hardware.

27.7.3.2   6.3.2 Complexity and Size

The code base is 6 million lines of code large, 30% of which are comments. It has a cyclomatic complexity of 3,299 of which 80% of the independent paths are in the *scripts* folder which houses the tests and sanity checks. The cognitive complexity results also bear the same ratio with the bulk of the complexity lying in the tests. Hence, we conclude that the bulk of code complexity is not in executing core functions of the operating system but in testing and building. Therefore, this complexity is not transferred to a user that ports Zephyr to their system and should satisfy hard real time complexity requirements.

27.7.4   6.4 Evolution Analysis

Zephyr was first released in May 2016 and by the time of writing, there are 14 releases. Despite steady and steep growth of the projects (from only supporting 15 boards in the first release, to 150+ boards in the 1.13 release), technical debt is being avoided rather than cured.
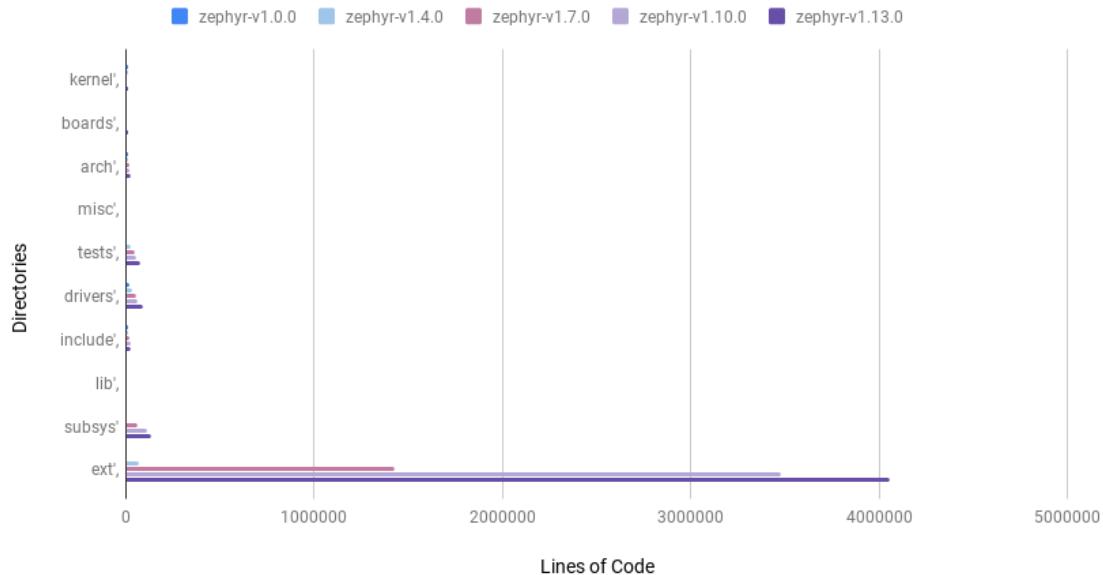
Figure 11a - Lines of Code evolution Chart

We analyzed the evolution of Zephyr by inspecting the growth of several components across releases. This was achieved using the tool `cloc`, counting the lines of code in selected top-level directories of the project: kernel, boards, arch, misc, tests, drivers, include, lib, subsys, ext. While C is not an object oriented language, we adopt terminology from [2] in describing its evolution.

Timeline (Releases of Zephyr)

Legend

NLOC = Normalised Lines of Code
Where lines of code are normalised to the
largest value of the file across releases

NLOC
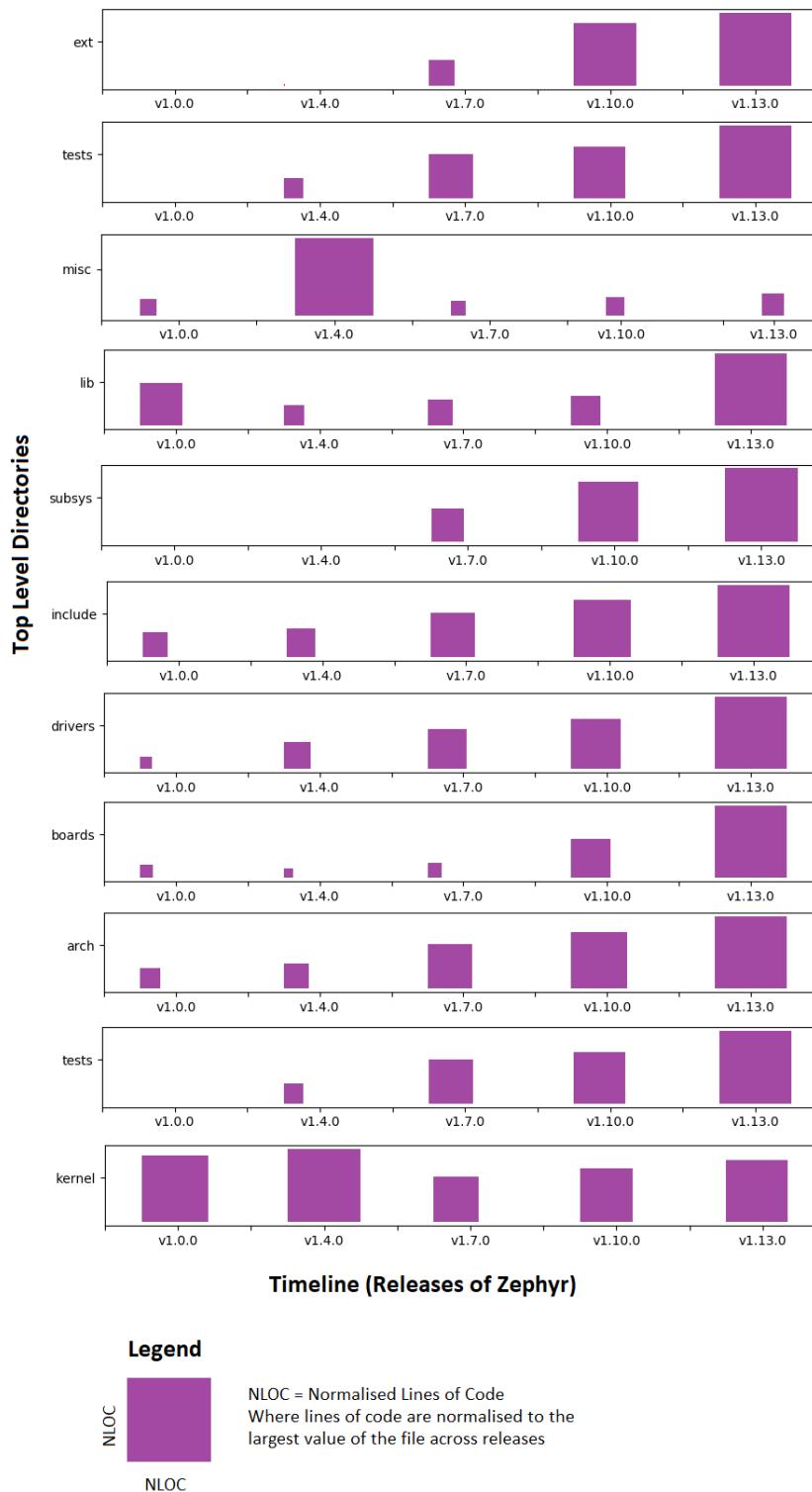
Figure 11b - Evo-

lution Matrix

The results, shown in Figure 11b, show *arch*, *boards*, *drivers* and *tests* as 'supernova' directories. Test code is seen to grow with core functionality and this is a good counter-measure against technical debt. We also observe the *misc* folder might be a technical debt indicator, as it contains code that does not fit into the folder organization otherwise. This is however a 'DayFly' directory which spiked once and was reduced in subsequent versions.
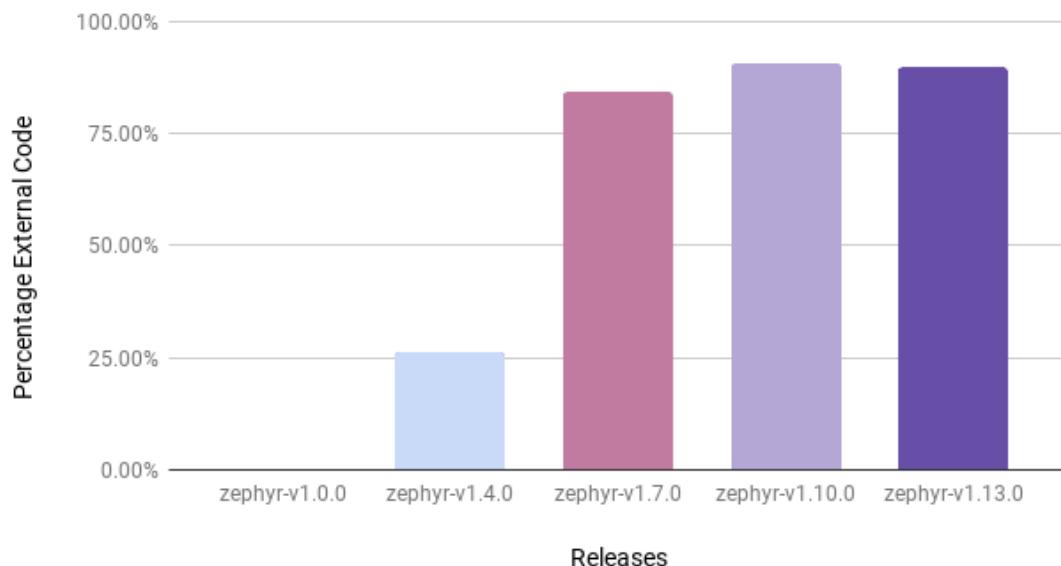

Figure 12 - Evolution of external code within the code base

The *ext* folder, while not a 'Persisitent' directory, has grown disproportionately with the rest of the project as shown in Figure 12. This contains external code integrated into the project and is currently the heaviest folder by the chosen LOC metric. Code contained is not in sync with changes and upgrades made by the original developers of these systems and thus, we identify this as technical debt. In the documentation, there are plans to move such code to separate repositories to keep externally maintained code away from the core Zephyr code. This will also ensure that such code remains in sync with the original systems.

27.7.5   6.5 Testing Debt

Technical debt in testing was evaluated by providing a measure of how much of the code is exercised in test scenarios. The Zephyr repository is linked to *Codecov*, an automated coverage tool that provides such results whenever continuous integration tests are run. This tool give the statistics as shown in Figure 13.
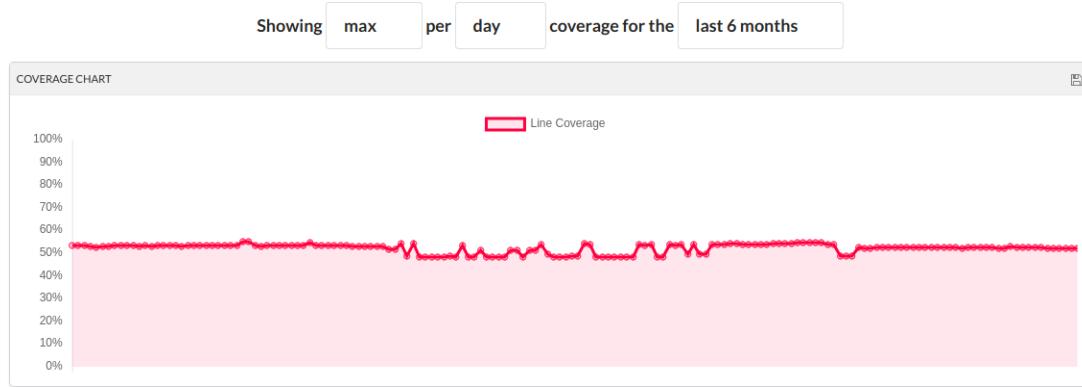
Figure 13 - Testing coverage chart

As seen in Figure 13, the coverage of tests on the repository has been at 50% over the past 6 months. This implies that no single day every portion of the code is tested. In Figure 14 below, we show a coverage report on a recent sanity check. It can be seen that only one directory under *arch/* and *boards/* was covered in the test.

Files	≡	●	●	●	Coverage
ext/hal/nordic/nrfx/hal	18	18	0	0	100.00%
soc/posix/inf_clock/soc.c	47	46	1	0	97.87%
kernel	2,008	1,753	148	107	87.30%
arch/posix	180	147	4	29	81.66%
include	1,308	1,050	91	167	80.27%
lib	2,488	1,862	304	322	74.83%
boards/posix	1,153	619	55	479	53.68%
subsys	37,214	17,700	4,119	15,395	47.56%
drivers	1,166	496	87	583	42.53%
Project Totals (309 files)	45,582	23,691	4,809	17,082	51.97%

Figure 14 - Test Coverage by File

Further investigation into tested aspects of the code show that this deficit exists because in normal use, the tests only runs for boards marked as default. As such, passing the sanity check only implies that the build works for a subset of the boards supported by the operating system. This is acknowledged in the docs and an option to run a more extensive test is provided, i.e., running the sanity check with an `--all` tag. To fix the possible loopholes in such a low test coverage, this should be made the default testing scenario.

27.8   7. Security Perspective

Security is an essential part of the Zephyr ecosystem. Aimed at IoT developers, the RTOS comes with an assurance of being one of the most secure options for application development. In this section we first analyze the threat model Zephyr faces followed by an outline of the well defined security process that Zephyr has adopted while also addressing security compliance requirements. This is mainly divided into Secure Development, Secure Design and Security Certification.

Figure 15 - Security Components

27.8.1   7.1 Threat Model

It is essential to first discuss an overview of the possible threats in the Zephyr project. Because Zephyr is an RTOS running on many different platforms and environments, it is susceptible to many different attack vectors. These can include malicious code submissions to the RTOS itself, exploiting communication protocol vulnerabilities of the deployed devices, etc. Besides attacks on the networking stack, functional security also needs to be addressed to prevent malicious or destructive behaviour. All the links shown in the context view can be a potential source of security risk. The figure below shows a simplified view of Zephyr's threat model, including attack vectors, sensitive assets and intentions.
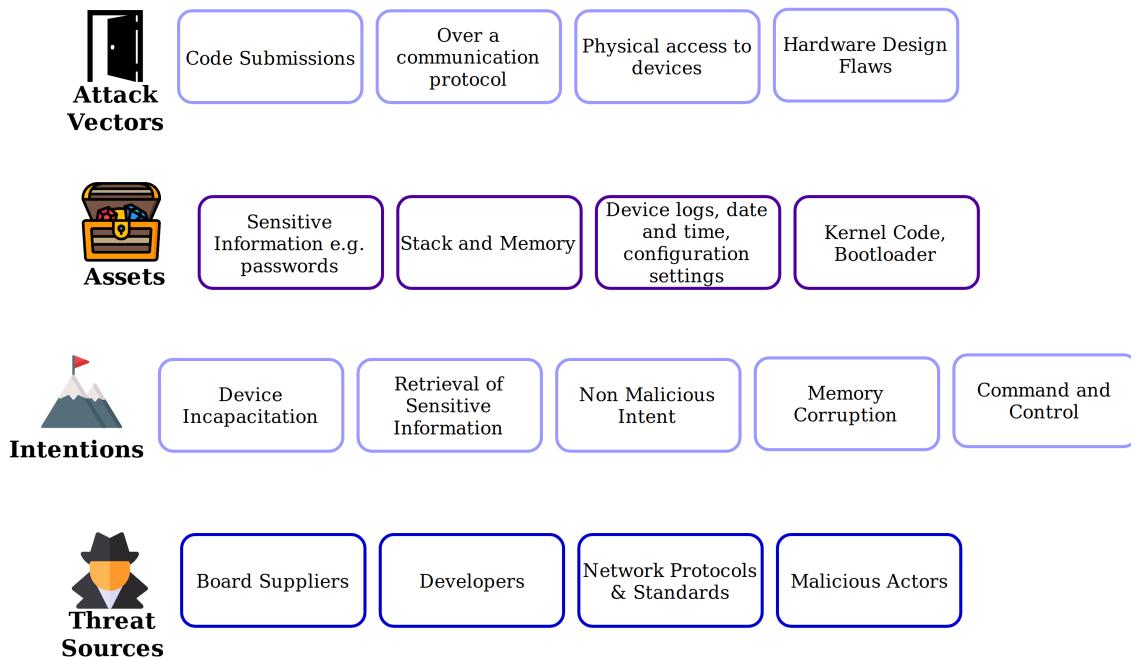
Figure 16 - Simplified Threat Model

27.8.2   7.2 Secure Development and Design

Zephyr provides a range of guidelines, tests and requirements to ensure functional security as well provide quality assurance. This is achieved by assuring code quality and correctness while additional security features, such as cryptographic libraries, are being developed as well. The Development process addresses security concerns by enforcing strict code reviews before any piece of code is merged to the master branch. This, in combination with static analysis tools ensures bug and backdoor-free software.

Security bugs, issue tracking and management is performed separately using JIRA where there has been some initial work on the definition of vulnerability categorization and mitigation processes. Security issues have more stringent reviews before they are closed as non issues (at least another person educated in security processes need to agree). They require security reviews to be performed by a security architect before each security-targeted release and each time a security-related module of the Zephyr project is changed. A security subcommittee has also been formed to develop the security process in more detail.

Secure coding guidelines, written by the security subcommittee are based on several design standards (for example "The protection of information in computer systems", by J. H. Saltzer and M. D. Schroeder) and address essential secure development principles such as: *designed system should be simple and small*, *process should operate with least privilege necessary*, etc.

The security architecture is based on a monolithic design and the Zephyr kernel and all of the applications are compiled into a single static binary which can be tailored during build-time. This static linking eliminates the potential for dynamically loading malicious code. It also provides *Stack protection* mechanisms to protect against stack overruns. In addition, applications can leverage *Thread separation* features which split

the system into privileged and unprivileged execution environments. *Memory protection* features enable partitioning of system resources (memory, peripheral address space, etc) again as a provision for secure design.

27.8.3   7.3 Security Certification

For certain stakeholders, certification of the code regarding safety and security is critical. Some users can only use certified software in their products and for developers and acquirers this means that certification can result in more widespread adoption.

The Zephyr project has announced that they plan to be the first open source RTOS to be (partly) certified. As certification of large projects is complex and costly, they aim for a step-by-step approach wherein first the core functionalities shall be certified. This includes the kernel, and some entities of the OS Services (mostly system services), as indicated in Module Organization. Selecting which parts of the code will be certified, and how the code needs to be adjusted for certification is the responsibility of the Security Working Group in collaboration with the Technical Steering Committee.

To make this possible, the release model will be adapted. This process is shown in Figure 17. The certified code-base will be a branch of the bi-annual long-term stable release (LTS). This branch will only contain a subset of the code base. This so-called auditable code base will be reviewed and adjusted to comply to certification standards, and then be submitted for certification. Important fixes for the master branch will be merged with the LTS and auditable code. The master, LTS and auditable code will be kept in sync, meaning that fixes for one branch will also be integrated in the other branches. One exception being new features, which will not be merged into the LTS and auditable code.
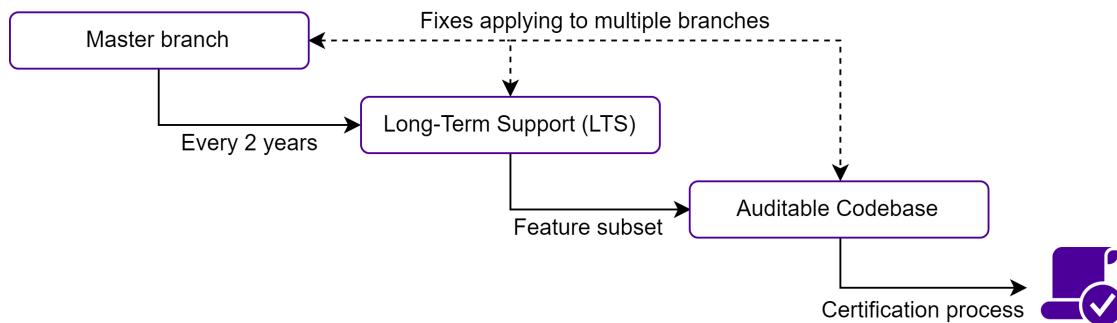

Figure 17 - Certification process

With this process, the Zephyr Project makes it possible for the OS to be (partly) certified in the future, making it more attractive for different types of users including those from industry. The project aims to acquire the following certifications: MISRA C:2012 (for safety, security, portability and reliability of embedded systems), IEC 61508 SIL 3 (for safety integrity) and ISO 26262 (for automotive certification).

27.9 8. Conclusion

Zephyr is one of the most promising open-source RTOSes available today. Despite being a relatively young player on the RTOS market, it is succesfull due to its well-defined organisational and development structure while also having a good understanding of what is important for IoT applications. Code quality is maintained by governing bodies overviewing the development process and reviewing the contributions thouroughly, which benefits not only Zephyr but also its community of developers.

First, the stakeholders involved in the Zephyr project were presented, showing its internal structure as well as its relation to the industry. By analyzing Zephyr's GitHub repository, it was observed that decisions for merging pull requests are based on experienced code reviewers, decision making bodies and automated tests. The development process was further analyzed in depth, describing their well-defined and organized git workflow, coding guidelines and the various tools available for developers. By giving an insight into the project's codeline and module organization, it became clear that code modularity is enforced as much as possible, and variability between different hardware platforms is managed through configuration files. Evaluating the codebase for technical debt resulted in interesting results and trends. These were mostly positive thanks to their adoption of good development practices. Lastly, the project was analayzed from a security perspective, outlining how the project guarantees security despite its open source nature and how they aim to be the first safety-certified RTOS.

In conclusion, Zephyr could serve as a role model not only for Real-time Operating Systems, but also for open-source software projects in general. It is a good example of how successful management of open-source contributions and a well defined and documented architecture can ensure and retain the software's quality.

27.10 9. References

[1] N. Rozanski and E. Woods, *Software Systems Architecture Working with Stakeholders Using Viewpoints and Perspectives*. Addison-Wesley, 2012.
[2] Lanza, Michele, *The evolution matrix: Recovering software evolution using software visualization techniques*. Proceedings of the 4th international workshop on principles of software evolution. ACM, 2001.
[3] Icons used in Figure 16 are made by Freepik from Flat Icon

27.11 10. Appendix

27.11.1 10.1 Pull Request Analysis

Pull Request	Status	Summary of Events
Request 1: add support for the Arduino Zero and the SAMD21 series	Merged	The aim of this request was to merge code that provided added support for two boards. These boards were already listed on the operating system but this patch fixed issues with board initialization and a few drivers. The timeline for the merge was 2 months which spoke to the significance of the request. There were 7 commits in total with ten changes requested from the time of request to the time of merge. This request fits into the larger roadmap plan of supporting as many boards as possible and this was the motivation/stakeholder interest. This request generated 147 conversation points and did not produce any code coverage difference.
Request 2:add Zephyr inline code generation with Python	Closed without merging	This request was closed without merging. Its aim was to add the feature of generating in line code. Such a feature is needed to dynamically configure of board pins from device tree information and automate repetitive coding tasks. The larger goal of this is to enable Zephyr be configured with device tree data. Discussions arising on this request included: -Concerns about some material used in the code that was originally licensed to MIT -Complications arising from using code generation in line with C code. 211 conversation points were raised in trying to resolve the issues arising and a decision was made that the project was not currently ready for such a change.
Request 3: uart: Add new asynchronous UART API (Transfer sub-API)	Merged	This pull request lifts the functionality of uart transfer from the programmer building on this operating system. Instead of writing direct interrupt handlers for this functionality, an API call is now used instead. Discussions on handling full duplex transfers and the efficiency of response to external events were raised by different developers. It took 3 months to merge this and this pull request has been referenced in two other pull requests that both aim to improve API functionality. This is directly a feature addition to the operating system and its impact will be most felt by users of the operating system.
Request 4: net: context: can: Fix typo in Kconfig option name	Merged	This was a high priority bug fix to a typo. The coverage was minimal and the decision to merge was obvious.

Pull Request	Status	Summary of Events
Request 5: Flash: Rework STM32F3 flash driver and configure F3 based boards for tests	Merged	This pull request was to improve board support for a class of boards who drivers had been lacking updates. Such commits fit into the project roadmap and as such there were no discussions into its usefulness. This qualified as an easy merge.
Request 6: board: stm32_min_dev: Add USB support	Open	This PR added USB support for a class of boards. This is still open as it is pending reviews even though it has passed all tests.
Request 7: Network logging overhaul	Merged	In February 2018, a new logging subsystem was proposed by @nashif, aiming to improve Zephyr logging capabilities with virtually no impact on performance, small memory footprint and user friendly interface. Few months later, @pizi-nordic provided a detailed introduction to the topic, high-level overview of the logger, requirements and some implementation concerns. As this presented a big overhaul of the system, the issue was divided on smaller parts and assigned among several developers. We analyzed the Networking part of the logger, which was assigned to @jukkar (Intel employee and member of the Zephyr project). His pull request re-implements logging parts of the networking subsystem, to conform to the new logging API. For better transitioning, both old and new ways of logging are supported, where the old API will be deprecated and finally removed in the future. Most of the discussion consisted of senior developers reviewing parts of the code. Perhaps the most noticeable critic is given by @pfalcon, where he points out the problem of too many commits in a single pull request, which makes it hard to review. Others commented on various syntactical inconsistencies, code clarity and redundant function calls. After many iterations of commits and reviews and approval of couple of senior developers, the pull request was merged into the master branch by @jukkar himself.
Request 8: Bluetooth: settings: Always initialize key when storing Client Features	Merged	A bug in the Bluetooth stack was noticed by @qbicz, where in certain conditions reading the Bluetooth configuration could crash the system. This issue was already fixed in previous release, but appeared again. Discussion between three developers revealed the possible source of problem -> new PR was created, which fixes the mentioned bug. The developer asked others to review if the changes affect other use cases. Upon several clarifications of coding decisions with @Laczen, and two approvals, the fix was merged into master by @nashif.

Pull Request	Status	Summary of Events
Request 9: doc: Rework Development Process section	Merged	This PR was filled by @nashif to provide updates on documentation about the development process of Zephyr. Despite having a leading role in Zephyr development, he asked more than three developers to review his changes. Upon replying to few questions asking for clarification on certain points, following comments were mostly about minor typos and style changes. Every successful update was reviewed again and only after few iterations of updates-feedback, the pull request was confirmed by all developers and merged to master branch. We observe the politeness and high-level of respect among developers, as most of the critiques where formed in a polite manner and started with gratification for the work that was being done.
Request 10: NVS: Non Volatile Storage solution for zephyr	Merged	This pull request provided a means to store configuration details of the operating system in non volatile memory of the boards and provide some persistence. The request was merged after changes were requested to its documentation and commit logs.
Request 11: net: zstream API to abstract socket transport protocols (e.g. TCP vs TLS)	Closed without merging	This pull request was targeted at a streaming API for socket based protocols but was closed without merging on the basis of its relevance to the project.
Request 12: drivers: sensors: Add lsm303dlhc driver	Closed without merging	This is a case where a pull request was superseded by another pull request. Both requests were to add support for a class of sensors in the operating system and the decision to close without merging was made not based on code quality or the necessity of the feature as we saw in other cases but simply because it was superseded.
Request 13: Gcov: Enable Code coverage reporting over UART	Merged	This pull request added a feature to enable users of the operating system generate code coverage reports over uart. The discussions leading to its successful merge focused on use cases and proper documentation of them for the users.
Request 14: net: sched: It's ok to preempt coop threads if they're polling	Closed without merging	This pull request proposed changes to the scheduling policy of the operating system. This is a very key aspect of any real time operating system and as such it raised a lot of discussion points. Such a change could alter the core functionality of the system and could imply deeper scheduling bugs. It was closed without merging for this reason.

Pull Request	Status	Summary of Events
Request 15: Introduce the west repository to the Zephyr Project	Merged	The west module(discussed under main isssues) is managed in a separate repository from the RTOS components of the project. This pull request introduced the tool to users and documented its cloning mechanisms. This would classify as an easy merge because the need for this documentation was clear-it is included in the project roadmap to include such a tool- and as such the only changes requested were document edits.
Request 16: doc: west: Document build system integration	Closed without merging	Similar to Request 15, this was a pull request for documentation on the west module. This time, not to introduce it to users but proper code documentation of its operation. This was however not merged but closed in favor of another pull request that achieved the same purpose. The combination of these requests paid an important technical debt.
Request 17: Drop ARP requests based on specific scenarios	Merged	This was an addition to the networking stack for the board to drop broadcast or multicast ARP requests as no single board really replies to such ARPs. This request was accepted because this is standard networking practice and was an oversight in the original implementation. It was however requested to log the reasons for any dropped packets to avoid loopholes in the future.
Request 18: power: Add device idle power management support	Open	This is an open pull request to provide a power management framework to suspend devices based on their idle state. It is still unmerged because of technical issues with its implementation.
Request 19: cmake: Introduce a user-local configuration file	Closed without merging	Closed without merging, this pull request aimed to provide a file for users to configure the operating system without having to run scripts. This was determined to remove the flexibility of switching between different toolchains and tools.
Request 20: Remove link layer reserve concept from network stack	Merged	This separates levels 2 and 3 of the network stack as implemented by the operating system. This change provided a 68% code coverage difference. Minor issues were noted and a merge was implemented after all the tests were passed.

27.11.2   10.2 Key Contact Persons

Name	Email Address	Role	Contact Made
Brett Preston	bpreston @ linuxfoundation.org	Program Manager for IoT Products at Linux. Currently assigned to LF Edge and Zephyr	None yet
Erwan Gouriou	erwan.gouriou @ linaro.org	Zephyr codeowner. Contributor for STM32 boards.	Reached out for guidance with submitting pull requests to the project

Chapter 28

Zulip

28.1 Abstract

Zulip is an open-source team chat application that provides all the benefits of real-time chat, distinguishing itself by remedying many shortcomings of common team chat applications through intuitive asynchronous means. The project is one of the larger open-source python projects and has extensive documentation and tools for programmers to start contributing. This report is an analysis of the architecture of the Zulip project. An overview of the project is provided by analyzing the project from both technical perspectives, such as the development process, technical debt and the deployment, as well as more organizational perspectives, such as the context of the project and the stakeholders involved. We find that the codebase of Zulip is of high quality overall and the project has a strong community, but there is a high dependence on one person, which has multiple risks tied to it.

28.2 Table of contents

28.3   1. Introduction

The concept of using team chats to improve the quality of communication and efficiency of teams within universities or companies has become increasingly popular in the last decade. The team chat application that most people have heard of and probably used is called Slack. According to the creators of Zulip, Slack and other applications following Slack's conversation model, such as Hipchat or Mattermost, are unable to provide the means to effectively communicate within organizations [1]. Zulip, "The worlds most productive team chat", offers an open-source alternative to these applications and attempts to remedy these shortcomings by focusing on asynchronous communication.

---

[1]Why Zulip? *Retrieved on 30-3-2019.* https://zulipchat.com/why-zulip/

Zulip is an open-source team chat application. The application was originally created by a small team from MIT and was acquired by Dropbox while the project was still in private beta. After 1.5 years of not being in active development, Dropbox decided to release the project as open-source software. Since then, Zulip steadily gained new contributors and eventually grew to be one of the largest and most active open-source projects on Github [2]. As of march 2019, Zulip has had merged more than 1600 pull requests created by the community of Zulip contributors, of which about 75 pull requests were merged in the past 3 months [3].

Zulip has a unique structure for their platform, which sets it apart from competitors. They argue that this structure is more efficient, less messy and has better persistence of information. An organization can create what is called a realm, which is comparable to a channel in Slack. These realms in turn contain streams with a list of topics, where a topic is much like an e-mail thread with a title. These topics are used to host conversations between users, which persist through long time intervals. This allows users to catch up to older conversations by filtering streams for topics,while still being able to converse in real-time. The figure below shows what this structure looks like in practice.

This chapter of the Delft Students on Software Architecture book 2019 aims to provide a clear overview of the Zulip project as a whole, shown from different perspectives. Not only the technical aspects of the project are highlighted, but also the organizational aspects. First, to understand the role of us as architects in the system, all the relevant stakeholders are identified and analyzed. Then, to gain a better understanding of the relationships and interactions in the system, a context view of the project is created. Once those are clear, we will go into the more technical aspects of the project by analyzing the process of the development of the project, as well as how to deploy the application. Lastly, we take a look at the technical debt of the project and finish with concluding remarks.

28.4 2. Stakeholder View

This chapter identifies the different stakeholders of Zulip and the decision-making process of Zulip is analysed to understand how the leaders of the project try to fulfil the needs of the stakeholders at all times.

28.4.1 2.1 Rozanski and Woods stakeholders

Type	Stakeholders	Description
Acquirers	Zulip team, Dropbox, Kandra Labs	The main acquirer is the core Zulip team, who manages the project. Dropbox acquired the project when it was still in private beta and later open-sourced it. Kandra Labs was founded by the lead developer of Zulip, Tim Abbott, to steward and financially sustain Zulip.

---

[2]Zulip History. *Retrieved on 23-03-2019*. https://zulipchat.com/history/

[3]Query on the Zulip Github pull requests page. *Retrieved on 29-3-2019*. https://github.com/zulip/zulip/pulls?page=3&q=is%3Apr+is%3Aclosed+is%3Amerged&utf8=%E2%9C%93

Type	Stakeholders	Description
Assessors	Zulip team, Tim Abbott	The role of assessors is taken by the Zulip team. They assess project's internal quality through code reviews and discussions. All merge requests are approved by Tim Abbott before merging into master branch.
Communicators	Zulip team, Zulip community	Most communication with stakeholders is done through Zulip community server, GitHub issues and documentation. Main participants are the Zulip team and community.
Developers	Zulip team, Zulip community	Developers of Zulip come from the community and everyone who wants to contribute. The Google Summer of Code and Google Code-In also contributed 30+ developers to the project.
Maintainers	Zulip team, Zulip community	Most of the maintenance is done through the documentation, managed by the community. The development environments are maintained by the core team.
Production engineers	Kandra Labs, Zulip Team	One of Kandra Labs products is a hosted Zulip service. The Zulip team is responsible for maintaining the production on-site.
Suppliers	i.e. Oracle, Hashicorp, Kubernetes	The project uses a multitude of available software packages and programs, which include (but are not limited to) Virtualbox, Vagrant and Kubernetes. This makes the creators of this software supplying stakeholders.
Support staff	Zulip team, Zulip community	The community acts as the support team. User support is provided through a help center which includes extensive documentation and contact e-mail address.
System administrators	Zulip team	Zulip has a hosted service for chat, administered by the core team.

Type	Stakeholders	Description
Testers	Zulip community	Testers are the same group as the developers. Testing is done by the community both through software testing and live testing using the community Zulip channel.
Users	i.e. Akamai, Wikimedia, LevelUp	Zulip is used for communication by other open source projects, Fortune 500 companies and various organizations.

28.4.2 2.2 Other types of stakeholders

Type	Stakeholders	Description
Competitors	Slack, Mattermost, others	Many companies offer similar services as Zulip. These parties hold a lot of interest in Zulip as a direct competitor, but have little to no power over it.
Partners	i.e. Hubot, Bitbucket, MailChimp	Zulip offers integrations for a multitude of existing applications.
Non-code contributors	Zulip community	Contributing to Zulip is not limited to writing code. Many members of the Zulip community contribute by reporting issues of, giving feedback on or translating the application.
Sponsors	USNSF, unnamed sources of funding	Kandra Labs received a large grant from the US National Science Foundation, along with additional sources of funding.

28.4.3 2.3 Power-interest grid

The acquirers and Tim Abott are the owners of the project and have full control of any alternations that are made to the source code. The community has a lot of interest in the project, which is their reason for contributing. Users, sponsors and competitors also have high interest in the project, as its quality influences people's choice of team chat application. The sponsoring stakeholder is mainly interested in the fact that their money is being put to good use. Suppliers hold both little interest and power over the project, as their products do not depend on Zulip, but rather the other way around. Lastly, partners have little interest in Zulip as their product is standalone and does not depend on Zulip, but hold some power, as one of the main qualities of Zulip is the integration with these tools.

28.4.4 2.4 Integrators

Zulip has one integrator, Tim Abbott. He reviews every single pull request and decides if a pull request gets merged. Other members of Zulip also review and provide feedback, but Tim Abbott always makes the final
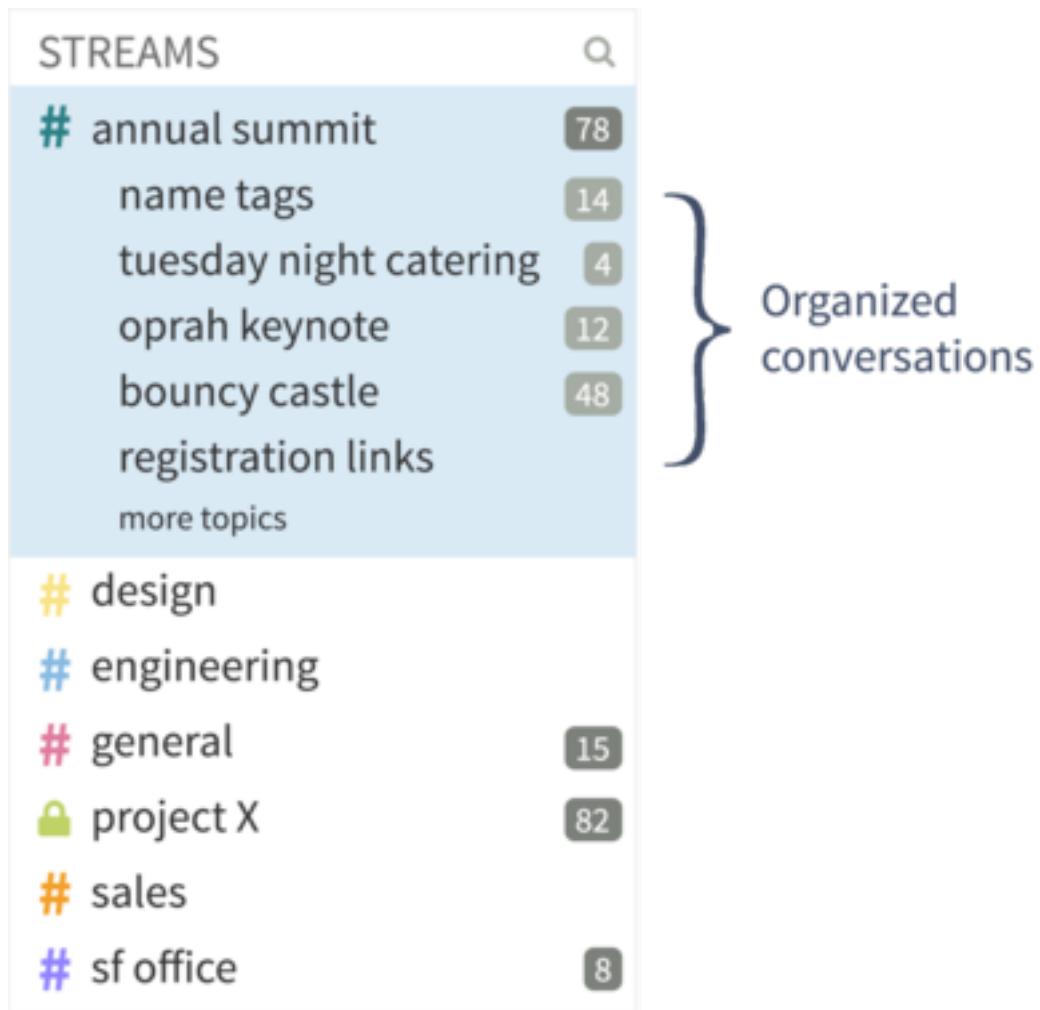
Figure 28.1: Example of a list of topics within a stream
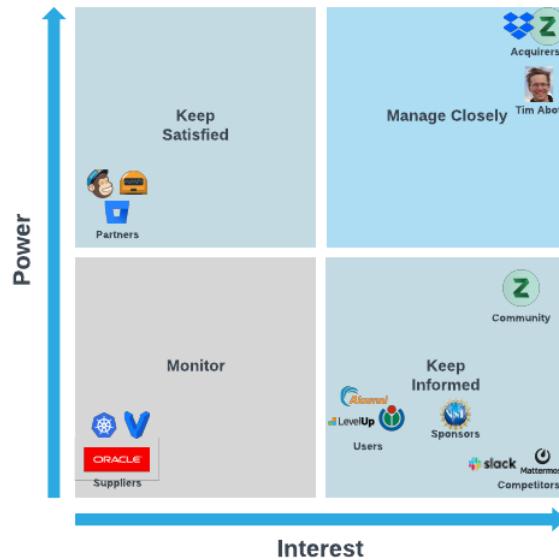
Figure 28.2: Power-grid diagram showing stakeholders involved with Zulip

decision.

Pull-based development model is used mainly for resolving issues and code reviews. Based on pull request analysis, features being implemented are often already discussed on a different platform. The pull request is then only used to review code for an already agreed feature.

Main reasons for rejecting/accepting pull requests are code quality/style and test passing. Tim Abbott is consistent in desired quality code, hence pull requests get merged only if quality code passes test cases.

Common issues that Tim Abbott faces are a high workload, difficult issues that nobody claims and developers that take a long time to incorporate feedback.

28.4.5 2.5 Relevant people

The leaders of this project are:

- [@Tim Abbott](https://github.com/timabbott): the absolute lead developer of this project.
- [@Rishi Gupta](https://github.com/rishig): focuses on product development.
- [@Steve Howell](https://github.com/showell): a full stack developer.
- [@Greg Price](https://github.com/gnprice): focuses on Zulip for mobile.
- [@Boris Yankov](https://github.com/borisyankov): focuses on Zulip for mobile.
- [@Akash Nimare](https://github.com/akashnimare): focuses on Zulip for desktop.

28.4.6   2.6 Contact persons

People that can be contacted on the Zulip channel for developers are: Tim Abbott, Rishi Gupta and Steve Howell. They are actively responding to (new) developers.

28.4.7   2.7 Decision-making process

The 10 most discussed accepted pull requests and the 10 most discussed rejected pull requests were analysed to reveal the decision-making process. Pull requests were selected by ordering them by the number of comments. It ensures that analyzed pull requests are relevant and relatively important.

Every accepted pull request, was only merged after extensive code review and after ensuring tested and functional quality code. The reasons for accepting the pull requests that we analysed can then be categorized in one of three ways: 1. It was a necessary bug fix. This was the case for PRs #7522 and #7472. 2. A new feature (#6012, #2828, #3026, #2688, #2911 and #2738). 3. Updated (or new) docs (#3056 and #2766).

The reasons for rejecting the pull requests that we analysed can also be categorized in one of three ways: 1. The implementation was never finished (#2881, #2771, #749). 2. The code was merged through other branches/PRs (#2552, #450, #6017 and #769). 3. The code was merged and pushed to master locally (#3923, #756, #3082).

Many of the analyzed unmerged pull requests were not actually rejected, rather they were merged through different pull requests/branches or Tim Abbott merged them into master on his local machine and then pushed this to master. When there is much discussion on a pull request, it has to be deemed important by lead developers and effort is dedicated to integrate it.

Most discussions on features happen on the Zulip chat channel and not on Github. The discussions on Github focus mainly on the code quality of a PR. Usually this is a lead developer doing a review of the code and the creator of the PR implementing this feedback.

Finally, Zulip created its own bot for Github. This bot does numerous things, including adding a tag indicating the size of a PR and sending a reminder to the creator of a PR when he has not responded to a review yet.

28.5   3. Context View

The context view of the system describes the relationships, dependencies, and interactions between the system and its environment [4]. This section describes the system's scope, responsibilities and its external entities to provide an overview of Zulip's environment.

---

[4]E. Woods, N. Rozanski (2012). Software Systems Architecture: Working with Stakeholders using Viewpoints and Perspectives. Addison-Wesley.
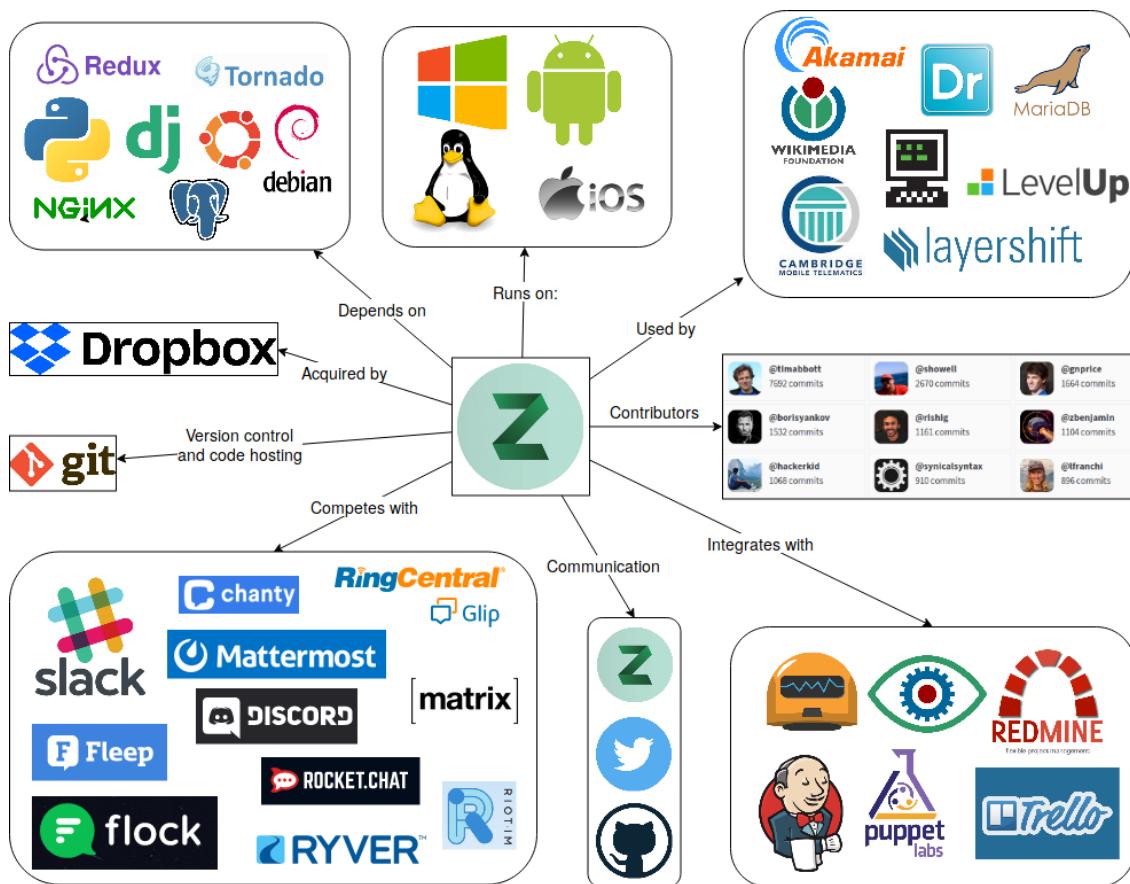
Figure 28.3: Context diagram showing dependencies and related external entities

28.5.1   3.1 System Scope and Responsibilities

Zulip is an open source group chat application, and so its main objective is to support various types of communication between team members [5].

The responsibilities of Zulip consist of:

- Processing messages
- The creation and maintenance of message streams, topics, user registrations and communities
- Provide native apps for Windows, Mac, Linux, iOS, and Android
- Support integration via APIs, webhooks and third-party extensions
- Support bot scripts
- Documented support on hosting Zulip on your own server

Zulip is a platform for communication between project team members, rather than a simple communication application. Modular design, open source licence and close relation with the community allows Zulip to grow based on needs of its users and clients. Furthermore, Zulip aims to become a hub for information management and development process monitoring, by integration with development tools and platforms.

28.5.2   3.2 External Entities and Interfaces

Zulip uses the following tools to manage its project:

- **Git**: version control
- **Github**: code hosting and issue tracking
- **Readthedocs.io**: documentation
- **Zulip**: community communication

In order to work, Zulip depends on Python 3.x and Django framework. These form basic building blocks of back-end services and facilitate communication between interconnected components. Back-end servers utilize Ubuntu and Debian software.

The core of Zulip makes use of widely used web technologies for server hosting, front-end rendering and communication between internal services.

As described in the context diagram, Zulip integrates with multiple third-party services. By providing interfaces and APIs for different platforms and programming languages, Zulip encourages the community to extend the list of available integrations by contributing own implementations. With these integrations being Zulip's major selling point, such dependencies on external and uncontrolled interfaces can cause a drawback. The more services are supported, the more coordination efforts between the integrator maintainer and service provider are needed.

Zulip has a strong community. From release date (Q3 of 2015) to March 2019, more than 350 people contributed to the project. A dedicated chat group is hosted to facilitate direct communications with contributors and users. Zulip also hosts multiple internships to support the community.

---

[5]Zulip documentation: Zulip overview. Retrieved on 20-02-2019. https://zulip.readthedocs.io/en/latest/overview/readme.html

28.5.2.1 3.2.1 Users and target audience

Zulip targets companies and organizations of any size as their primary users. Main focus at this point is on programming community, due to heavy integration with platforms and tooling used in this industry. The audience for Zulip is by no means restricted because of this. With its recent success and growing interest in the community, new features and integration with less development-oriented tooling will broaden the target market for Zulip.

28.5.2.2 3.2.2 Competition

As messaging and communication form the basis of the Internet's functionality, the market for professional communication platforms is strongly saturated. Slack being the market leader for many years is an obvious competitor to Zulip, however many other organizations joined in an attempt to provide the functionality Slack is missing. Multiple open source initiatives offer similar communication experience, while allowing self-hosting, higher customizability, or providing integration with external tooling. The strongest open-source competitors are Mattermost and Rocket.Chat.

28.5.3 3.3 Impact of the System on its Environment

Zulip introduces advanced functionality, user-friendliness, open and active community, and levels of communication management which are lacking in competing products. Because of this, the market will see certain degree of new dynamics.

Considering the level of Zulip's integration with many development platforms and tools, these can benefit from exposure to new users, and consolidating their functionality into communication streams of users. The unique selling point of Zulip is its integrations with tools, which other products adopt to if they want to compete.

28.6 4. Development View

This perspective considers the structural design of the project, the development process and configuration management, as well as various system constraints and standards. The goal is to gain a better understanding of the architecture supporting the development of Zulip.

28.6.1 4.1 Module Structure

The code structure of Zulip is organized into directories, which broadly represent the components. As can be noticed in the diagram below, some of the subdirectories contain various assets for related functionality, without clear separation of asset types. For example, testing code is spread amongst various components, with additional documentation and scripting resources. This was done to cater for the open-source development model, and ensure focus on relevant components by assigned developers.
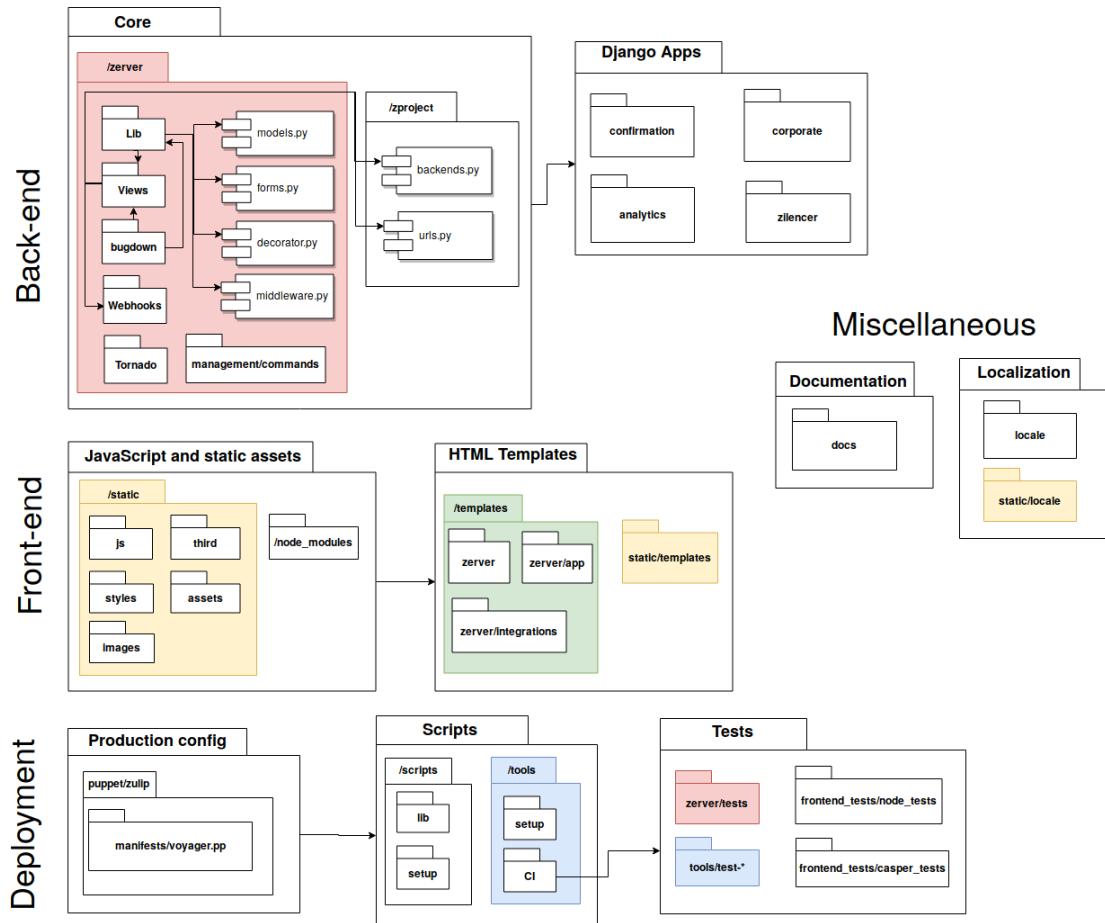
Figure 28.4: Diagram depicting Zulip's module structure. A functional categorization for separate pipelines (back-end, front-end, deployment and miscellaneous) is presented with the directories holding the source code for the involved modules.

28.6.1.1   4.1.1 Back-end

The core of the application consists of classes holding server functionality. These are responsible for user account creation, management and authentication, message processing, notification system and various queues, database content management, and handling of Zulip realms, groups and streams. Most of communication and redirection functionality is also defined here.

Many classes belonging to the core modules are responsible for supplying graphical components with appropriately formatted and parsed data. Zulip team has also created their own implementation of Markdown (called *'Bugdown'*) for custom rendering of messages and ensuring only desired content will reach the end user.

Core library classes are over-saturated with responsibilities and do not provide clear separation of duties. A good example of it is the 'actions.py' module located in the '/lib' directory. The class has over 5500 lines of code responsible for notification events, realm configuration, user updates, and message processing.

Server-side of the application can be customized according to an organization's needs through extendable Django modules. These enable organization-oriented customization of the service, without having to alter the server code-base.

This layer depends heavily on third-party implementation of server-side software. *Django* and *Tornado* software packages serve most of this module's functionality and any major change to their APIs will necessitate a significant amount of migration and maintenance costs.

28.6.1.2   4.1.2. Front-end

The front-end application and various related assets are contained within the *'/static'* and *'/templates'* directories. Zulip uses Handlebars templates for live-rendering of interface elements. The front-end is also capable of validating correctness of rendered messages and confirming this against the back-end server. Simple user messages are processed locally on the client-side, unless they require advanced validation (e.g., in case of source code boxes, or Markdown scripts in a message). This effectively acts as a load-balancer for rendering tasks.

Due to a lot of custom code in JavaScript and HTML templates, this layer bears many third-party dependencies on *npm* packages for HTML manipulation, graphics handling and input parsing.

28.6.1.3   4.1.3. Deployment

The Zulip project uses Puppet as their main approach to deployment. Puppet is self-described as "an automatic way to inspect, deliver, operate and future-proof all of your infrastructure and deliver all of your applications faster" [6]. To run a standard standalone Zulip server, the main Puppet manifest (Puppet programs are called manifests) called *voyager.pp* is used to manage all the modules that are needed for an entire Zulip installation. Lastly, scripts that might need to ran manually, for instance *restart-server*, are contained in the scripts module of the repository.

---

[6]How Puppet Works. *Retrieved on 20-3-2019*. https://puppet.com/products/how-puppet-works

Third-party dependencies come from this layer's heavy reliance on testing suites for front-end (CasperJS and TravisCI) and back-end (CircleCI). The testing environment also requires an existing Vagrant installation to run.

28.6.1.4   4.1.4. Integration with external modules and bots

One of the biggest advantages of Zulip over its competition is the ability to integrate the communication application with external services and tools. These event-driven integrators and automated bots are extensions, as they are not part of the main code-base. They react to internal or external triggers by posting messages. They are special user types, and are maintained in separate repositories.

28.6.2   4.2 Common Design Model

Strict coding style and code review process is in place, to ensure consistency and ease of code-base understanding, despite multitude of collaborating developers. Event-driven functionality is standardized through use of RabbitMQ software and specified queueing strategies. Creation of graphical components of the application is guided by a set of HTML templates to maintain coherent style among all user interface views. Rendering of messages and many textual elements is done by custom Markdown implementation, further driving consistent design. A variety of linters and automated code quality verification tools are used, to safeguard uniformity of coding style and detect most common programming mistakes.

A dedicated chapter in the documentation is devoted to contributors joining the development cycle, describing style, requirements and constraints on the code quality [7]. A tutorial on the process of including a new Zulip feature is provided [8], explaining the dependencies and most important source code which the developer should consider, while extending functionality of the application.

28.6.3   4.3 Development Process

While there have been many releases of Zulip, the project is still in active development. With over 40 releases, roughly 500 contributors and more than 500 merged commits in the past month (the month of February 2019), a solid development process is of great importance.

28.6.3.1   4.3.1 Version Control

Zulip uses Github as the main tool for version control. The conventions that are used on this platform are described below.

---

[7]Zulip documentation: Code contribution guide. *Retrieved on 04-04-2019*. https://zulip.readthedocs.io/en/latest/contributing/index. html

[8]Zulip documentation: Writing a new application feature. *Retrieved on 04-04-2019*. https://zulip.readthedocs.io/en/latest/tutorials/ new-feature-tutorial.html

28.6.3.1.1 4.3.1.1 Issues Any contribution that is made to Zulip is first introduced as an issue, usually created by one of the members of the core Zulip team. These issues are labelled with descriptions of the project area that the issue relates to (testing-coverage, production etc.), the status of the issue (in progress, inactive etc.) and for which kind of developer the issue is fit (e.g. good-first-issue). Zulip employs a bot to provide possible contributors with information on the issue, such as external references to the issue and tips for first-time contributors.

28.6.3.1.2 4.3.1.2 Pull Requests Pull requests are the main way for a contributor to receive feedback on their work on a specific issue. Zulip encourages submitting work-in-progress pull requests as early and often as possible. This way, they aim to manage expectations of contributors correctly and prevent unfinished work from being merged.

28.6.3.1.3 4.3.1.3 Releases So far there have been many releases of Zulip. The project adheres to a checklist that distinguishes between a week before the release, right before the release, during the release and post-release. Summarizing, these include upgrading dependencies, generating and extensively testing a release tarball, publishing the new version on external platforms and making announcements and pushing the release commit to the master branch respectively.

28.6.3.2 4.3.2 Testing

Development package contains a dedicated 'Vagrant' environment for easy test deployment. This facilitates quality of testing results and reduces the configuration time needed by developers for setup.

Additionally, Zulip makes use of continuous integration tools. Two pipelines have been created:

- *CircleCI*, for running front-end and back-end tests on a variety of Linux distributions,
- *TravisCI*, for exclusively running end-to-end production installer tests.

28.6.4 4.4 Documentation and localization

Main documentation files are stored in the *'/docs'* folder, however many text files are scattered around the project in locations corresponding to what they relate to. This goes in line with the main idea of compartmentalization by relevance.

Localization assets and main documentation are residing in dedicated directories, together with necessary scripts and assets for compiling complete documentation.

28.7 5. Deployment View

The deployment view highlights aspects of the system which come to light after it has been built, when it transitions to live operation [9]. According to Rozanski and Woods (2012) this "is useful for any information system with a required deployment environment that is not immediately obvious to all of the interested

---

[9]E. Woods, N. Rozanski (2012). Software Systems Architecture: Working with Stakeholders using Viewpoints and Perspectives. Addison-Wesley.

stakeholders." In Zulip's case this view is important for clients that want to host their own Zulip servers, since they need to know how to do this. This section identifies the key aspects of the deployment of Zulip.

28.7.1   5.1. Concerns

This section discusses the requirements of the runtime platform, third parties that Zulip depends on, and network requirements. The information in this section is retrieved from the documentation of Zulip [10].

28.7.1.1   5.1.1 Runtime Platform Requirements

Zulip's server software is designed to run on Linux, supporting versions 16.04 and 18.04 of Ubuntu strain, and Debian 9. These particular operating systems have no hardware constraints, and this allows Zulip to be deployed on general purpose computing nodes. Use of a dedicated machine is advised, to avoid version incompatibility for libraries on which multiple software applications depend. It is possible to install Zulip on a system with 2GB of RAM, but 4GB of RAM is recommended for an organization with more than 100 users, and for more than 1000 users the server should have 8GB of RAM. The server should have at least 10GB of free disk space.

Zulip uses Postgres as a database engine. It can run on the same machine as the primary server or on a remote one. A remote database is recommended for isolation. For the database server SSD disks are highly recommended for faster reading and writing. With 30GB of RAM and 8 cores the database should be able to scale to 10,000s of active users. File storage can be handled locally, however for scaling purposes and use of multiple Zulip servers, it is recommended to use Amazon S3 as file storage solution.

Zulip is designed to run behind a reverse proxy server (e.g. Nginx). This can be useful to re-route traffic and/or can act as an additional security layer. The reverse proxy server can be setup on the same machine as the primary server. The Zulip server will then never have to be exposed to the public internet.

28.7.1.2   5.1.2. Third party dependencies

The Zulip server software depends on numerous third party dependencies. They include:

- **Postgres**: required as a database engine.
- **Redis**: an in-memory data structure store, used for caching.
- **Memcached**: another caching system.
- **RabbitMQ**: message-queueing software.
- **Django**: the actual application server.
- **Tornado**: a real-time push server, used for events.

28.7.1.3   5.1.3. Network requirements

The server needs to handle incoming HTTPS access (usually over port 443) from the network where the users reside. Communication over unencrypted channel (HTTP) will be redirected automatically to HTTPS. To

---

[10]Zulip documentation: Development environment. *Retrieved on 01-04-2019.* https://zulip.readthedocs.io/en/latest/development/index.html

facilitate secure communication the server needs an SSL certificate for the domain name it uses. Furthermore, outgoing HTTP(S) access to the public internet is also required. Finally, outgoing SMTP access to a SMTP server is necessary for Zulip to send and receive email.

28.7.2   5.2. Runtime platform model

The runtime platform model shows an overview of all systems and nodes that are involved when Zulip is running. It is a schematic overview of the aforementioned runtime platform requirements, third party dependencies and network requirements. Parts of the diagram are based on an architectural overview in the Zulip documentation [10].

28.8   6. Technical Debt

Technical debt reflects the implied cost of additional work that needs to be done in the future caused by choosing an easy solution now instead of using a better approach that would take longer [11]. This implies that technical debt is incurred by choice. This section discusses the technical debt of Zulip.

28.8.1   6.1. Identification of technical debt

We used SonarQube to inspect the quality of the code. Bugs, vulnerabilities and code smells can be detected by automated analysis. Therefore we assume that a developer that wrote such faulty code knew about it but decided not to fix it. This makes it technical debt. Code duplication is also technical debt, as updating this code requires changes in multiple places. The following table shows the results of the analysis by SonarQube.

Programming language	Lines of Code	Bugs	Vulnerabilities	Code smells	Duplicated code
JavaScript	36,000	18	0	18	0.7%
Python	92,000	3	0	400	1.4%

Overview of the metrics found by SonarQube

Most of the code smells found by SonarQube in the JavaScript code are expressions that always evaluate to `true`. These issues are simple to resolve. Manual analysis of the other code smells show that these are also easily solvable. SonarQube categorizes 6 bugs as "Critical". All 6 are a function call where the caller provides one argument too many. This can be solved by removing the redundant argument. SonarQube estimates that it takes around 5 hours to fix all code smells and bugs related to JavaScript.

For the Python code there are only 3 bugs which does not impose a lot of technical debt. However, there are many code smells. A lot of the code smells are related to the cognitive complexity of functions and the number of parameters that functions take. These are issues that require some thought about the architecture of the code to be resolved, and thus impose a higher technical debt. SonarQube estimates it takes 9 days to resolve all code smells and bugs. An important note is that the code smells only cover 0.2% of Python code,
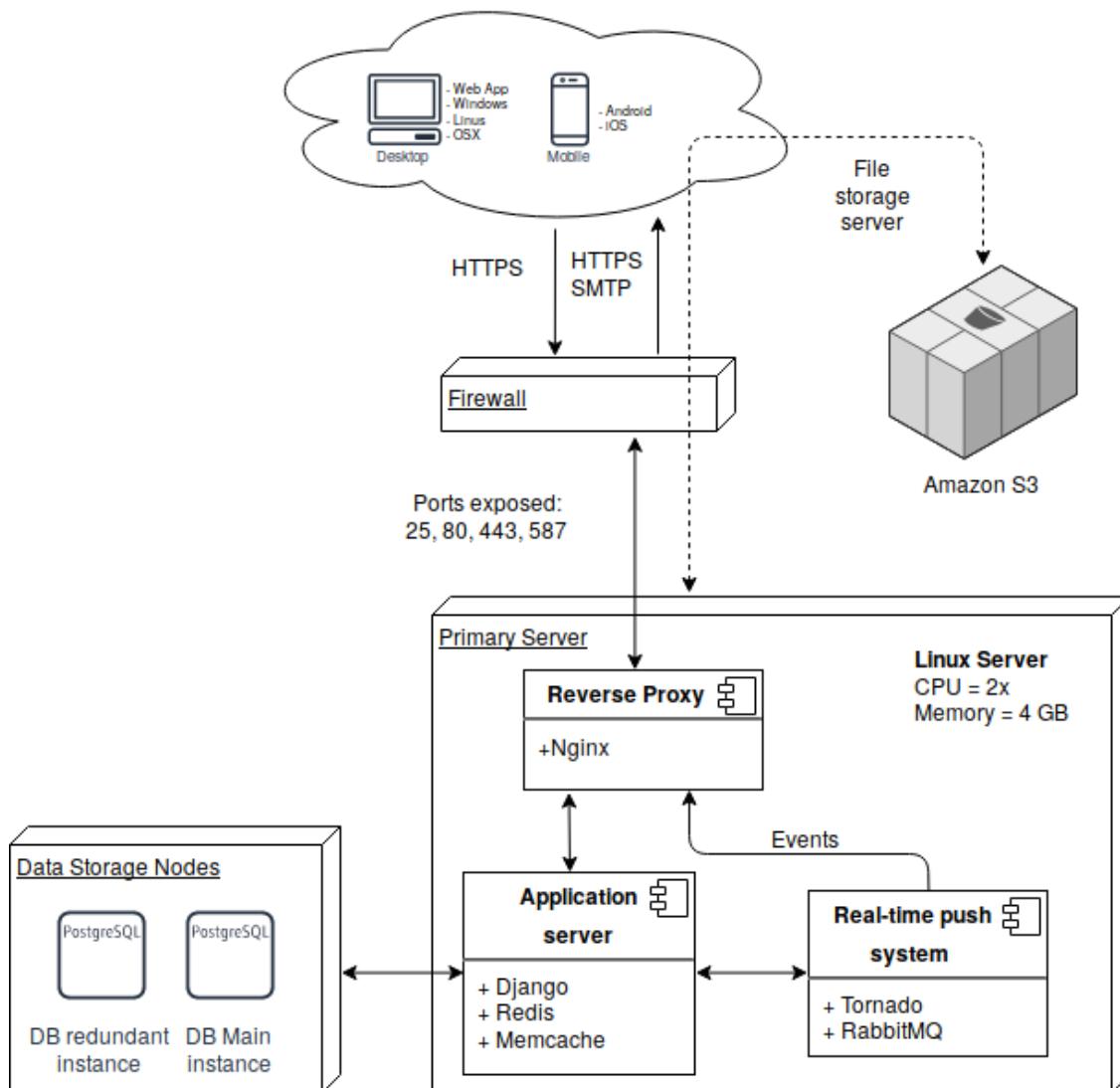
Figure 28.5: Graph showing an abstraction of a runtime platform model of Zulip's components.

so the majority of Python code is 'clean'.

28.8.2   6.2. Discussions about technical debt

We searched for keywords in the code that represent technical debt, namely `todo`, `fixme` and `hack`. These keywords imply technical debt since it indicates that the author knew the code is not optimal but decided not to resolve it immediately. The table below shows the amount of these instances.

Language	TODO	FIXME	HACK
JavaScript	135	8	22
Python	262	2	59

The count of different keywords in the codebase

Technical debt in Zulip can also be discussed through issues on Github and/or Zulip's own chat realm for developers. However, Zulip does not record technical debt explicitly outside the code. This could be improved by adding issues with a dedicated label for every piece of code that incurs technical debt.

28.8.3   6.3. Evolution of technical debt

To assess the evolution of technical debt in Zulip we ran SonarQube on multiple previous releases of Zulip. We picked the following releases: - 1.1.5: the first release on Github, November 2013. - 1.2.0: February 2014, shortly after this there was no development on the project for a long time. - 1.3.0: February 2017. The first release again after Tim Abbott picked up the project. - 1.4.0: February 2017. - 1.5.0: February 2017. - 1.6.0: June 2017. - 1.7.0: October 2017. - 1.8.0: April 2018. - 1.9.0: November 2018. - 2.0.0: the release discussed above, which is the current release.

28.8.3.1   6.3.1 Evolution of bugs

Over time the JavaScript codebase grew from 12,000 to 36,000 lines of code (LOC). Despite this growth, the number of bugs in JavaScript only increased slightly over time. This indicates that the linters and code reviews are working well to keep the bugs out of the JavaScript code.

The Python codebase grew from 32,000 to 92,000 LOC and also has few bugs currently. There is a big increase followed by a big decrease in bugs. It is not clear what caused the increase, but the decrease is most likely caused by the migration to Python 3 in release 1.7.0. A lot of the bugs in the 2 releases before that were most likely not bugs in older versions of Python, but SonarQube just scans the project assuming it uses Python 3. Zulip has very little bugs, so it seems that here the linters and code reviews are also working well.

28.8.3.2   6.3.2 Evolution of code smells

In release 1.4.0 there were 57 code smells and in release 1.5.0 7 were left, indicating a big cleanup was done in between.
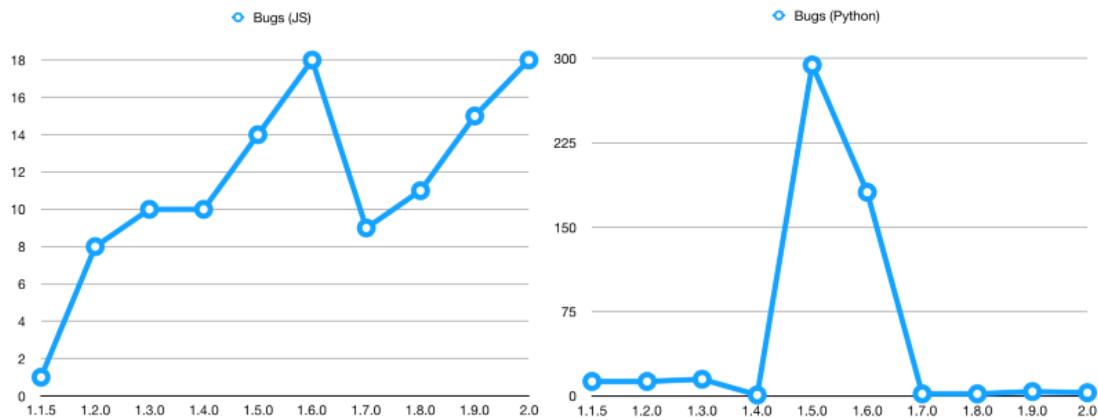
Figure 28.6: The progression of the number of bugs in the JavaScript and Python code of Zulip, respectively
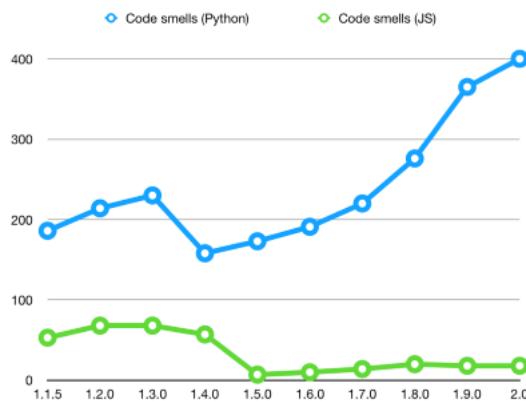

Figure 28.7: The progression of the number of code smells in the code base of Zulip

As for the Python code the code smells steadily increased over time. It seems that with the addition of so much code, it is inevitable to get some code smells.

28.8.4   6.4. Testing debt

Zulip uses Codecov to measure code coverage of their Python testing suite. The maintainers have set certain rules regarding the required test coverage for pull requests to be submitted, to keep testing debt small. The initial line coverage requirement to run a successful build is low (50%), to allow for quick addition of features. However when a PR changes a pre-existing source code file, that file must not decrease in test coverage. This holds for both Python and JavaScript source code.

28.8.4.1   6.4.1. JavaScript code coverage

The coverage output of the JavaScript code is shown below.   This was the output after running `./tools/run-js-with-node --coverage`.

Directory	Statements	Branches	Functions	Lines
static/js	67.17%	59.57%	66.64%	67.15%

Coverage output of JavaScript code

The majority of the JavaScript source files have 100% statement coverage. However some files have very low coverage, such as `subs.js`. Most coverage issues are related to user interface interaction, which are tested using the tool CasperJS. CasperJS performs automated browser testing [12]. It is hard to tell what the actual code covered by CasperJS is, as only contains assumptions of the structure of the DOM output, and does not test the intermediate calls in the JavaScript code. Overall the testing debt for JavaScript code is low as it has high coverage, but it is difficult to measure the debt for the uncovered code.

28.8.4.2   6.4.2. Python code coverage

The overall Python line coverage is 22%, which is a lot lower than the required 50%. The tool `test-backend` declares a list of files that do not have proper coverage yet, with an explanation per file, and are not taken into consideration when calculating the overall coverage. Remarkable is the fact that some major lib files are included here, while a comment says `Major lib files should have 100% coverage` as these are components with many dependants. This results in high testing debt.

28.8.5   6.5. Documentation debt

Another form of technical debt is documentation debt. The code of Zulip is poorly commented which makes it difficult for new developers to get started on the project. Updating the codebase to have sufficient comments would be an immense task and therefore imposes a big technical debt on the project.

28.8.6   6.6. Impact of technical debt

The metrics found by SonarQube and the keywords described in section 6.2 show that there is relatively little technical debt in the codebase of Zulip. The biggest issue is the lack of documentation of the code. There are few comments in the code which makes it difficult for new developers to start developing. Because adding comments to the project is such a big task, it will likely never be done.

28.9   7. Conclusion

Zulip is a very active open source project on Github. Every day multiple issues and pull requests are submitted and features are being added weekly. However, this is a double edged sword, as such activity poses large challenges in regard to maintenance. Currently Tim Abott, the main developer of the project, decides for every pull request whether it gets merged or not. Such dependence on one person creates high risk for a project.

The community of Zulip is very welcoming to new contributors. They provide a Zulip chat channel for new contributors and a specific label to mark issues to be solveable for new contributors. They set rules on a CI for code coverage, git usage, static analysis, cross-platform testing and more to keep code quality high.

Overall the codebase of Zulip is of high quality. There is a low amount of code smells and bugs, however it should contain more comments. The project is under very active development, and has a strong community, but there is high dependence on one person (Tim Abott) to keep it up-to-date.

28.10   References