IN4135:
Software Architecture

Arie van Deursen, February 2020

https://se.ewi.tudelft.nl/delftswa/2020/index.html

Learning Objectives

What do you expect / hope to learn in the
TU Delft Software Architecture Course?

Architecture in Context

- **What** problem will the system solve?
  - What are the key capabilities and properties that will generate value?
  - In what context will the system operate?
  - How will this context evolve the upcoming years?
- **How** should the system be structured?
  - So that it realizes the required capabilities and properties
  - Now and in the future
- **Who** will create and evolve the system?
  - Which team roles are to be distinguished?
  - How do the people in the teams cooperate?

The Architecture of a System (IEEE Definition)

- The set of fundamental concepts or properties
- of the system in its environment,
- embodied in its elements and relationships,
- and the principles of its design and evolution.

Reliable Knowledge

What is reliable knowledge in software architecture?

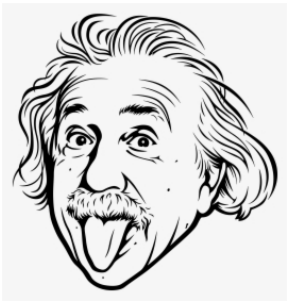
What does it take to increase our knowledge on software architecture?

- Systems that we know how to build
- Systems that we have not yet built, but know the process
- Systems we dream about

-- Grady Booch

Post Positivism

- **Conjectures and Refutations:**
  The growth of scientific knowledge
- Testing of hypotheses
- A priori use of theory



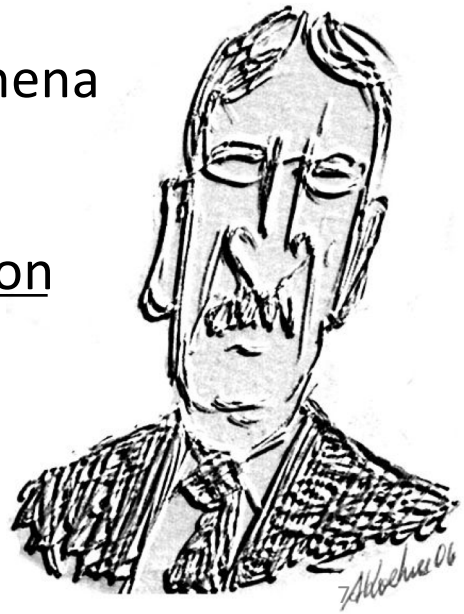
Not the dominant paradigm in (software) engineering

Pragmatism

- Clarify meanings of intellectual concepts by tracing out their "conceivable practical consequences" (Charles Pierce, 1905)

- Pragmatism ... does not insist upon antecedent phenomena but upon consequent phenomena;

  Not upon the precedents but on the possibilities of action (John Dewey, 1931)



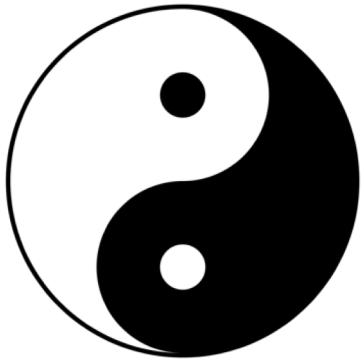
Knowledge Sources

- **Theory:**
  - Battle-tested knowledge that has proven effective in the past
  - Frameworks, patterns, processes, ...
- **Practice / Experience**
  - Engineers who learned from their own experience
  - Engineers who reflect on their own and others way of working
- **Research**
  - Data collection, hypotheses generation, process support ...
  - At the boundaries of what we presently know (... and possibly wrong)

Learning About Architecture

1. Just do it: Engage in architectural activities in realistic setting
2. Study / *internalize* existing theories and approaches
3. Confront the two with each other
   - How does this theory really work?
   - Does this theory apply to my system? Why? Why not?

Thesis:	Theory
Anti-Thesis:	Practice
Synthesis:	Understanding



Learning Theories through Practice!

- Choose an open source system, from GitHub
- Study its architecture in depth
- Study to what extent architectural theories apply

Top open source projects by contributors

This year, popular open source projects are topping 10K contributors. Two have been on this list since 2016: microsoft/vscode and ansible/ansible. New in 2019 are flutter/flutter, firstcontributions/first-contributions, and home-assistant/home-assistant.*

NUMBER OF CONTRIBUTORS TO OPEN SOURCE PROJECTS

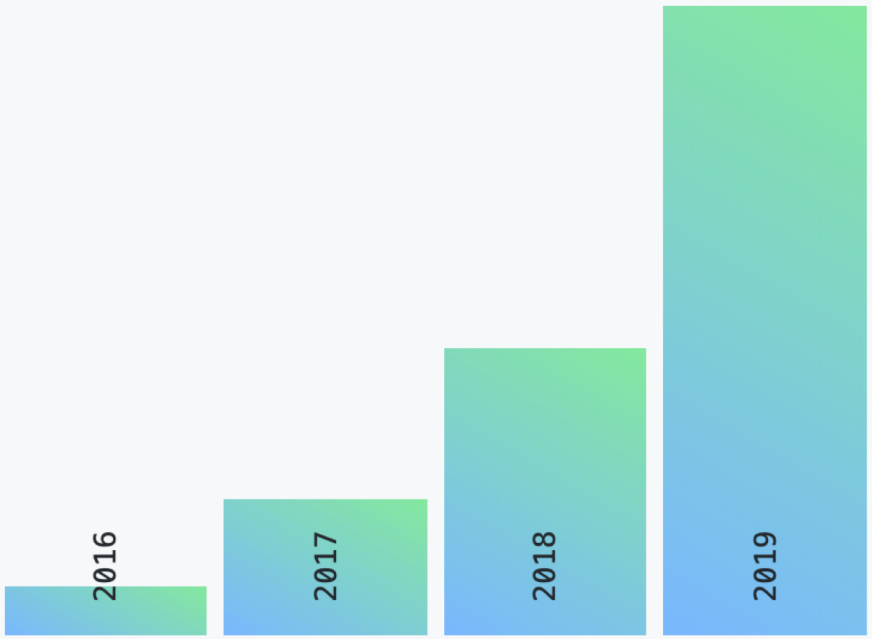
01	microsoft/vscode	19.1k
02	MicrosoftDocs/azure-docs	14k
03	flutter/flutter	13k
04	firstcontributions/first-contributions	11.6k
05	tensorflow/tensorflow	9.9k
06	facebook/react-native	9.1k
07	kubernetes/kubernetes	6.9k
08	DefinitelyTyped/DefinitelyTyped	6.9k
09	ansible/ansible	6.8k
10	home-assistant/home-assistant	6.3k

Fastest growing open source projects by contributors

Kits and frameworks for building apps and websites across languages and platforms are seeing contributor growth this year. Since its 1.0 release in December 2018, flutter/flutter has climbed to #2.*

CHANGE IN CONTRIBUTIONS TO OPEN SOURCE PROJECTS

01	aspnet/AspNetCore	**346%**
02	flutter/flutter	**279%**
03	MicrosoftDocs/vsts-docs	**264%**
04	istio/istio	**194%**
05	aws-amplify/amplify-js	**188%**
06	helm/charts	**184%**
07	ValveSoftware/Proton	**182%**
08	gatsbyjs/gatsby	**179%**
09	storybookjs/storybook	**178%**
10	cypress-io/cypress	**178%**



Growth of Jupyter Notebooks, 2016-2019

How else can we tell data science is growing on GitHub? The use of Jupyter Notebooks (by number of repositories with Jupyter as their primary language) has seen more than 100% growth year-over-year for the last three years.*

Software Architecture = Team Work

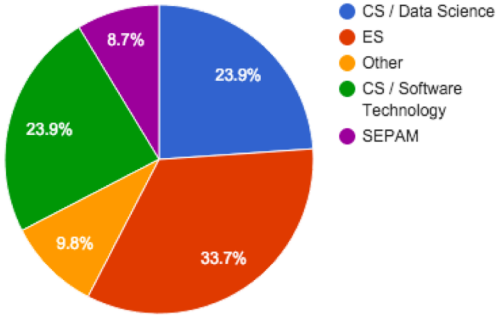
- Few systems are built by a single person
- The architecture is a means of communicating important design decisions

- Therefore, all assignments done in teams.

- Teams of 4 (4 != 3, 4 != 5, 4 != 2+2)

Team Composition

- Aim for diversity
  - Git knowledge
  - Programming language expertise
  - Bachelor background
  - Current master (track)



- Brightspace discussion forum “Partners Wanted”
- Form your group on Brightspace (Collaboration / Groups)
- **DEADLINE: Monday February 17, 17:00**

Assignment 0.1: Team Selects Project!

- Could in principle be any project
- *You should be excited about it*
- Project should be under active development (> 1 pull requests / day)
- Project should be sufficiently complex (not docs only, e.g.)
- Project should be open to outsiders (check CONTRIBUTING.md)
- Project should not be used years 2018 or 2019 (check DESOSA)
- Use Brightspace "Claim your project" forum.
- **DEADLINE: Monday February 17, 17:00**

Generated list of ~40 projects on course page

Assignment 0.2: Manage your Time!

- Considerable freedom (own initiative) in *what* you do
- You need to *explain* how you spent your time
- 5 EC = 140 hours; In 8 week course = 17.5 hours per week!
- Per student: short, reflective journal, commit one entry per week
  - Track how many hours *you* spent
  - Main activities conducted
  - Main output produced
  - Summary of key things learned

Open Learning & Learning from your Peers

- This course is open by design
  - You learn from what others are doing
  - You share your work with others

- Work-in-progress writing is visible within course only

- Your interaction with open source systems is public
- You can decide if you want to make your writings public

All Communication: *Mattermost*

- https://mattermost.ewi.tudelft.nl/sa-2019-2020/

- Town Hall – main channel
- Off-Topic – your random noise
- Team-XYZ (public)
  - Main communication hub for your team
  - Accessible to all; others can help / learn

- *Use mattermost to*
  - *leave an evidence trail of you work.*
  - *Integrate with other teams*

See registration link on BrightSpace

https://medium.com/@daniel.heller/ten-principles-for-growth-69015e08c35b

Daniel Gebler
@daniel_gebler

Ten Principles for #Growth as an #Engineer:

1. Reason about business value
2. Unblock yourself
3. Take initiative
4. Improve your writing
5. Own project management
6. Own education
7. Master tools
8. Communicate proactively
9. Collaborate
10. Be reliable

4. **Improve your writing**: Crisp technical writing eases collaboration and greatly improves your ability to persuade, inform, and teach. Remember who your audience is and what they know, write clearly and concisely, and almost always include a tl;dr above the fold.

Arie van Deursen @avandeursen · Dec 17, 2019

Jeff Bezos on the importance and difficulty of clear writing:

"We don't do PowerPoint (or any other slide-oriented) presentations at Amazon. Instead, we write narratively structured six-page memos. We silently read one at the beginning of each meeting."

sec.gov/Archives/edgar...

2 4 13

“Not surprisingly, the quality of these memos varies widely. Some have the clarity of angels singing. They are brilliant and thoughtful and set up the meeting for high-quality discussion. Sometimes they come in at the other end of the spectrum.”

“The great memos are written and re-written, shared with colleagues who are asked to improve the work, set aside for a couple of days, and then edited again with a fresh mind. They simply can’t be done in a day or two.”

"Surely to write a world class memo, you have to be an extremely skilled writer? [...] Even in the example of writing a six-page memo, that's teamwork. Someone on the team needs to have the skill, but it doesn't have to be you."

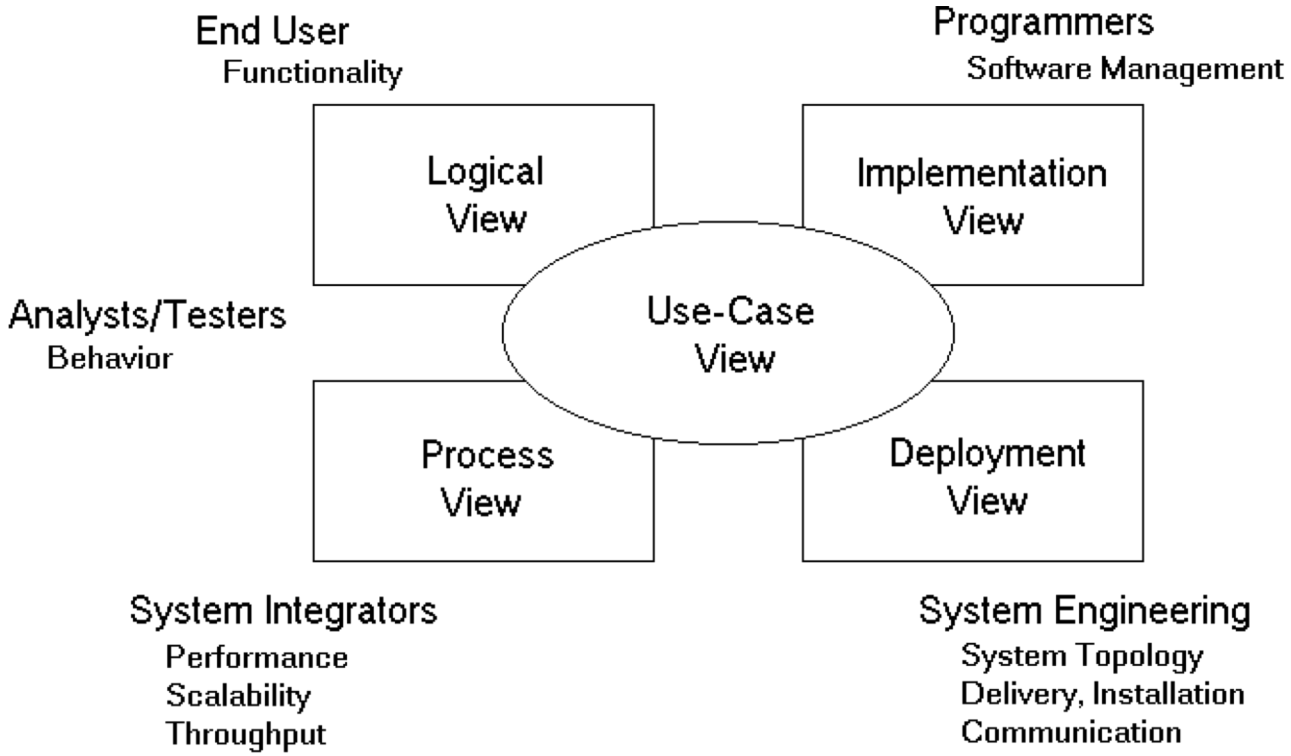
https://100x.engineering/the-power-of-the-narrative/

Assignments E1-E4: (Technical) Essay Writing

Each team writes four essays (1000-1500 words):

1. the product vision, including required capabilities, roadmap, product context, and stakeholder analysis.
2. architectural decisions made, including system decomposition, tradeoff points, as well as architectural styles and patterns adopted.
3. quality control and assessment; and
4. a deeper analysis based on the lectures or other relevant material specific to the system of choice;

Kruchten's "4+1 Views"

IEEE Software, November 1995

[Intermezzo: E3 Quality Assessment with SIG]

The Delta Maintainability Model: Measuring Maintainability of Fine-Grained Code Changes

Marco di Biase*†, Ayushi Rastogi†, Magiel Bruntink*, Arie van Deursen†
*Software Improvement Group - Amsterdam, The Netherlands †Delft University of Technology - Delft, The Netherlands
Email: *[m.dibiase, m.bruntink]@sig.eu, †[a.rastogi, Arie.vanDeursen]@tudelft.nl

***Abstract*—Existing maintainability models are used to identify technical debt of software systems. Targeting entire codebases, such models lack the ability to determine shortcomings of smaller, fine-grained changes. This paper proposes a new maintainability model – the Delta Maintainability Model (DMM) – to measure fine-grained code changes, such as commits, by adapting and extending the SIG Maintainability Model. DMM categorizes changed lines of code into low and high risk, and then uses the proportion of low risk change to calculate a delta score. The goal of the DMM is twofold: first, producing meaningful and actionable scores; second, compare and rank the maintainability of fine-grained modifications.**

We report on an initial study of the model, with the goal of understanding if the adapted measurements from the SIG Maintainability Model suit the fine-grained scope of the DMM. In a manual inspection process for 100 commits, 67 cases matched the expert judgment. Furthermore, we report an exploratory empirical study on a data set of DMM scores on 3,017 issue-fixing commits of four open source and four closed source systems. Results show that the scores of DMM can be used to compare and rank commits, providing developers with a means to do root cause analysis on activities that impacted maintainability and, thus, address technical debt at a finer granularity.

I. Introduction

Software maintainability defines the ease with which software can be modified, *e.g.*, to correct defects, or to improve its functionality [1]. Measuring and improving maintainability helps managing technical debt, a concept introduced by Cunningham [2] and then refined by Fowler [3], [4]. To grasp technical debt, Ernst *et al.* [5] noted that code analysis and measurements are the main concrete methods used to understand technical debt.

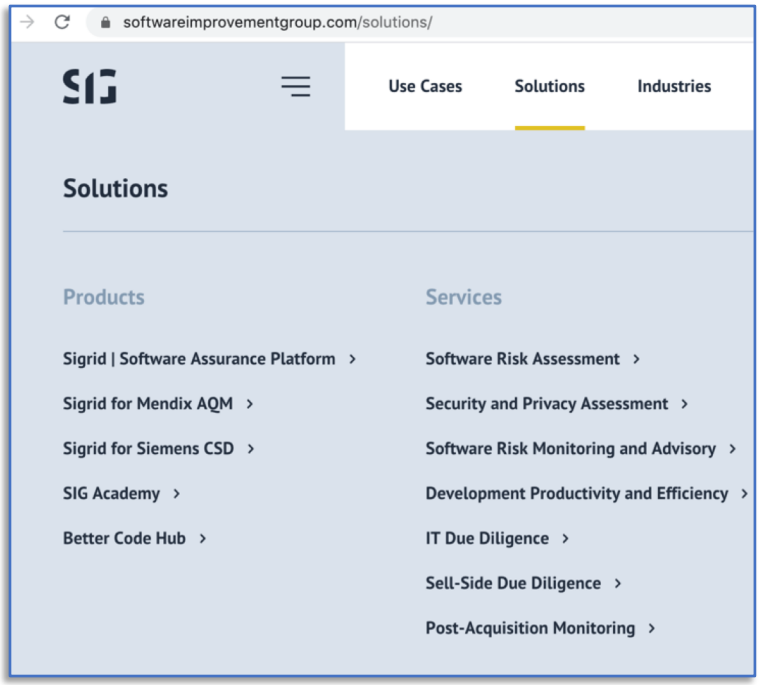
Maintainability is a complex concept, and past research shows efforts towards data-driven approaches to indicate maintainability, with several models proposed [6]. Some of these models are in active use in research or in industry, such as the SQALE method [7], the Software Improvement Group Maintainability Model (SIG-MM) by Heitlager *et al.* [8], or the QUAMOCO model [9]. These models take the source code of complete software systems at fixed moments in time (*i.e.*, snapshots) as their primary units of maintainability analysis. Such an analysis matches well with the relatively low update and release frequencies of larger (legacy) software systems.

In recent years however, development is driven by an increasing level of change granularity. Smaller changes are being implemented in software releases [10], continuous integration tools ensure that every committed code change is integrated into the primary line of development [11], and developers use fine-grained mechanisms such as pull requests to review and accept code changes [12]. To support these modern development practices, maintainability [illegible] models need to take fine-grained code changes ([illegible] or limited number of commits) as their prim[illegible] analysis. In this scenario, developers could addr[illegible] that creates technical debt at a finer level of abs[illegible] promptly use actionable suggestions rather than re[illegible] portions of code. In fact, nonfunctional requirem[illegible] maintainability are a concern during the modern [illegible] processes because they contribute to the compl[illegible] tures [13] and managing technical debt repercu[illegible] project resources [5].

Despite measuring differences in code metrics ([illegible] McCabe Complexity [14]) provide a superficial i[illegible] fine-grained change-maintainability, they lack th[illegible] formalized models available for complete softw[illegible] Such models, though, are not suitable to measure [illegible] code maintainability as they carry the burden [illegible] system codebase. In our view, a fine-grained ma[illegible] model should rely on code measurements able t[illegible] amount of risk introduced over the total change[illegible] meaningfully report the maintainability of fine-g[illegible] changes abstracting from the full system codebas[illegible] scoring system that allows result comparison as[illegible] system it is applied.

In this paper we propose a new model for t[illegible] which we named the Delta Maintainability Mo[illegible] The DMM measures the maintainability of a co[illegible] ratio between low risk code and the overall code m[illegible] DMM identifies code riskiness by reusing software [illegible] risk profiles of the SIG-MM, while applying new [illegible] and scoring for software metric deltas at the le[illegible] grained code changes like commits or pull requ[illegible] of aggregating at the system level. The DMM does not aim to replace the SIG-MM, rather, complement its system-level analysis with direct support for fine-grained code changes.

In the study, we first focus on understanding if the SIG-MM measurements suit the fine-grained scope of the DMM. In a manual inspection process, we measure a sample of 100 code changes against the intuition and expertise of the main authors. Then, we analyze data of the DMM's outcomes on 3,017 fine-grained code changes, originating from four open-source and four closed-source systems, to understand whether the score produced suits the needs of ranking and interpret-

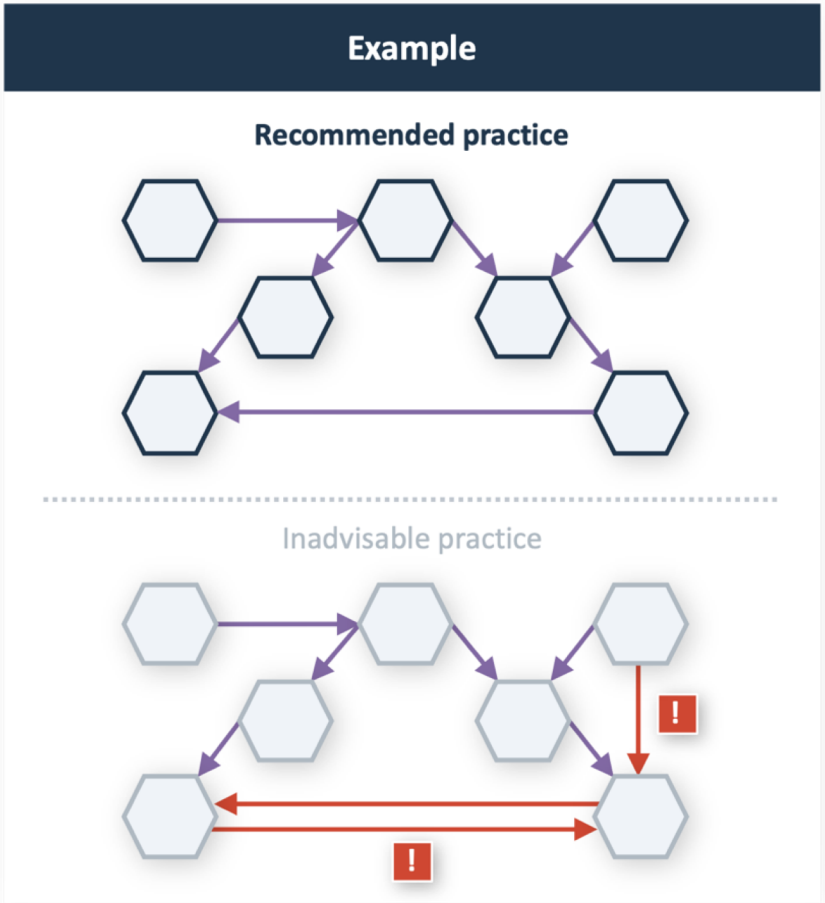


Componentization of a Software System

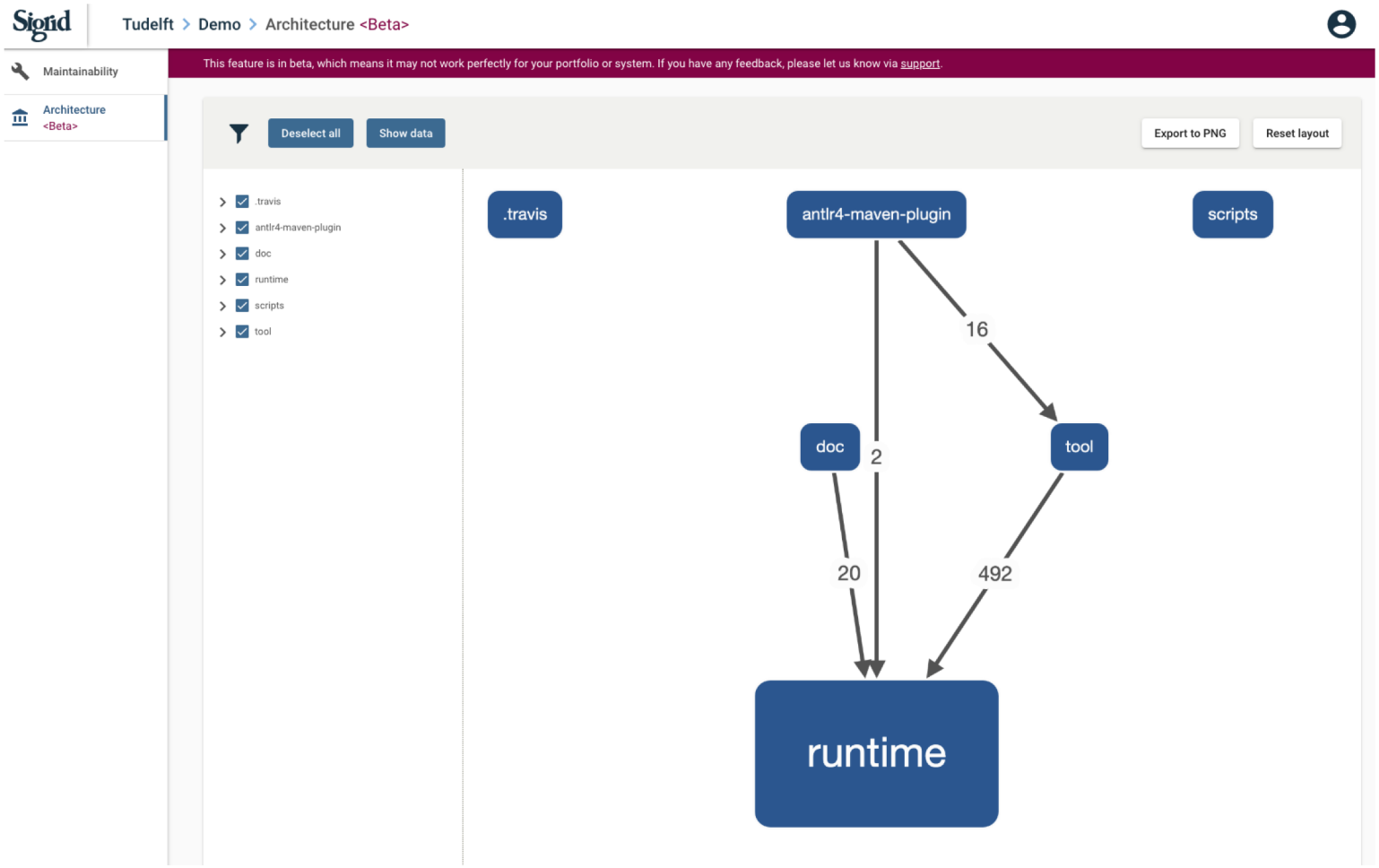
- Important to measure related architectural aspects that I will explain and share with you in 2 weeks (Lecture of February 26)
- Useful for the remainder phases/assignments of your project
- Will help you answering questions like:
  - How does each component interact with each other?
  - Is this supposed to be the case?
  - Can we identify possible improvements or can we learn something?

Why you need to do it?

Component



Why you need to do it?



Need help?

- I’ll be sharing a guideline on how to identify system components
- Discuss first within your team and then try to draft a first componentization
- Still stuck? -> #system-component in Mattermost
- A good idea is to join the channel so that you learn from other projects and you can help each other
- We will have ~25 projects, so we cannot pinpoint ALL your issues because of time reasons

Assignments E1-E4: (Technical) Essay Writing

Each team writes four essays (1000-1500 words):

1. the product vision, including required capabilities, roadmap, product context, and stakeholder analysis.
2. architectural decisions made, including system decomposition, tradeoff points, as well as architectural styles and patterns adopted.
3. quality control and assessment; and
4. a deeper analysis based on the lectures or other relevant material specific to the system of choice;

General Essay Requirements

- Sentences, paragraphs, and sections are coherent. They naturally build upon each other and work towards a clear message.
- The arguments are sound, and of adequate technical depth.
- The English writing is grammatically correct
- The text clearly references any sources it builds upon
- The essay has a voice of its own, and is original in its approach
- It is independently readable
- It uses meaningful and appealing images or infographics.

Public Writing makes Better Writers

- Objective 1: Write for the course
- Objective 2: *Write for the world*

- Throughout the course, your team can make your work available

- *Delft Students on Software Architecture* (DESOSA)

se.ewi.tudelft.nl/desosa2019/

Delft Students On Software Architecture

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.

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.

First-Hand Experience

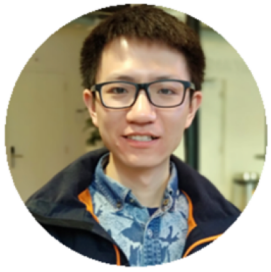
Last but not least, all students made a substantial effort to try to contribute to the actual projects. With these contributions the students had the ability to interact with the community; they often discussed with other developers and architects of the systems. This provided them insights in the architectural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

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

delftswa.gitbooks.io/desosa2018/mattermost/chapter.html

Published with GitBook

Mattermost®


Shang Xiang
@CoolTomatos


Xinyue Wang
@XinyueWang94


Luke Prananta
@lmikaellukerad


Jasper van Esveld
@GitHubJasper

Abstract

Mattermost is an open source, cloud-based and self-hosted Slack alternative. The company came to realize the limitations and restrictions of Slack when they adopted it as their messaging service in 2014. This is when they decided to build their own messaging software, Mattermost. In this chapter we analyze the software architecture of Mattermost by looking at the stakeholders, the context view, development view, deployment view, security perspective and technical debt as defined by Rozanksi and Woods [1]. We find out that Mattermost has a well-organized development process and is currently in the process of repaying technical debt by moving the web app to Redux.


delftswa.gitbooks.io/desosa-2017/content/neovim/chapter.html
Introduction
Arduino
Gradle
JabRef
JUnit5
Jupyter Notebook
Kafka
Kibana
Magento
Mapbox GL JS
Matplotlib
Mockito
Neovim
Netty
Node
Processing
Scikit-learn
Scrapy
Syncthing
Telegram Web
VSCode
Neovim
Team
neovim
In order of appearance: Ioannis Petros Samiotis, Thomas Millross, Sander Bosma and Jente Hidskes.
Abstract
This chapter describes the software architecture of Neovim: an open source code editor based on Vim. This analytical essay will provide interested readers with objective and relevant insights into the challenges and architectural decisions of the Neovim development effort. Neovim's software architecture is assessed within the Rozanski and Woods [1] framework. The system stakeholders are detailed and categorised. Then the context and development viewpoints are described, followed by an analysis from
https://delftswa.gitbooks.io/desosa-2017/content/neovim/chapter.html

DESOSA: Past / Present

Past

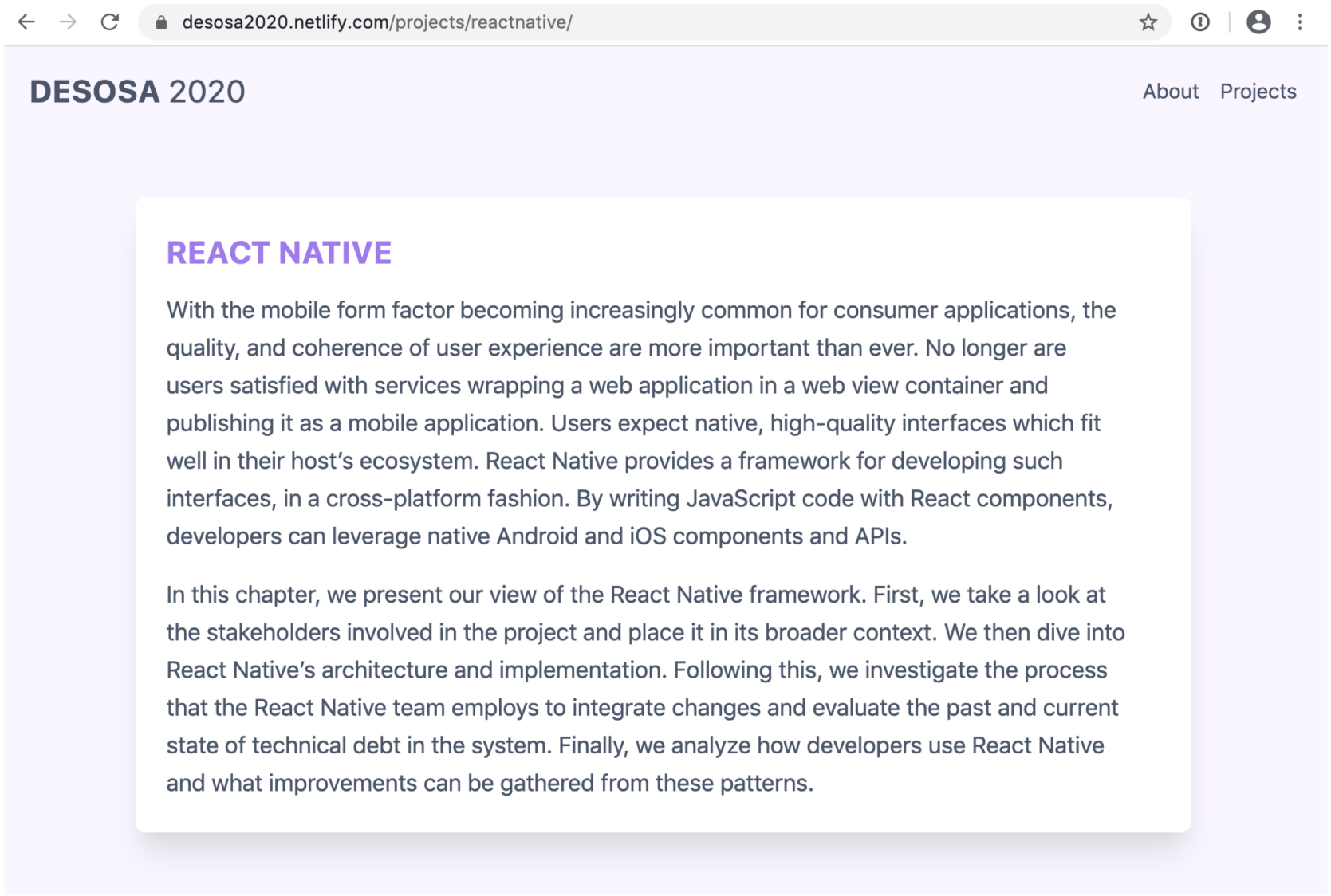
- Book with chapters
- One chapter per team
- Each team own git repo
- Book published *after* course
- Graded by teachers

- Teams can opt-out
- All documents in markdown

Present

- Collection of essays (blogs)
- Four essays per team
- All teams in one shared git repo
- Blogs shared *during* course
- Peer review

- Teams can opt-out
- All documents in markdown


desosa2020.netlify.com/projects/reactnative/
DESOSA 2020
About
Projects
REACT NATIVE
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.

How React Native is developed

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.

reactnative

Feb 11, 2020

Read more »

The people behind React Native

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.

1. Nick Rozanski and Eoin Woods. Software Systems Architecture: Working with Stakeholders using Viewpoints and Perspectives. Addison-Wesley, 2012. ↩

reactnative

Feb 9, 2020

Read more »

IN4315 > 2019-2020 > desosa2020 > **Repository**

master desosa2020 / projects / reactnative / _posts Lock History Find file Web IDE

/ +

Add example content with images
Casper Boone authored 6 hours ago

Unverified 1abf2044

Name	Last commit	Last update
..		
images/reactnative	Add example content with images	6 hours ago
2020-02-09-stakeholders.md	Use consistent markdown file extension	10 hours ago
2020-02-10-people.md	Use consistent markdown file extension	10 hours ago
2020-02-11-how-react-native-is-develope...	Add example content with images	6 hours ago
2020-02-11-react-native-pull-request-anal...	Add example content with images	6 hours ago

Learn from Open Source Architects: *Offer them a Contribution*

- Make a useful contribution to the system you study
- Offer it to the system's architects as a *pull request*
- They will discuss it with you, … and hopefully *merge* it.

Get in touch with the architects!

Make them read your work

Interview them for your blog?!

Contributions in Earlier Years

- Around 3-4 pull requests per team made
  - Around half of them merged

- Refactorings, bug fixes, documentation, features
  - Docker, Jekyll, JUnit5, yarn, …

- Start small (“micro-contribution”) and EARLY
- Be professional, polite and efficient
- Follow development guidelines (CONTRIBUTING.md)

GitHub, Inc. [US] https://github.com/docker/docker/blob/master/CONTRIBUTING.md

Branch: master ▾ **docker / CONTRIBUTING.md** Find file | Copy path

unclejack CONTRIBUTING: add guidelines regarding email 64e8fa9 on Jan 8

42 contributors and others

435 lines (319 sloc) | 17.6 KB Raw | Blame | History

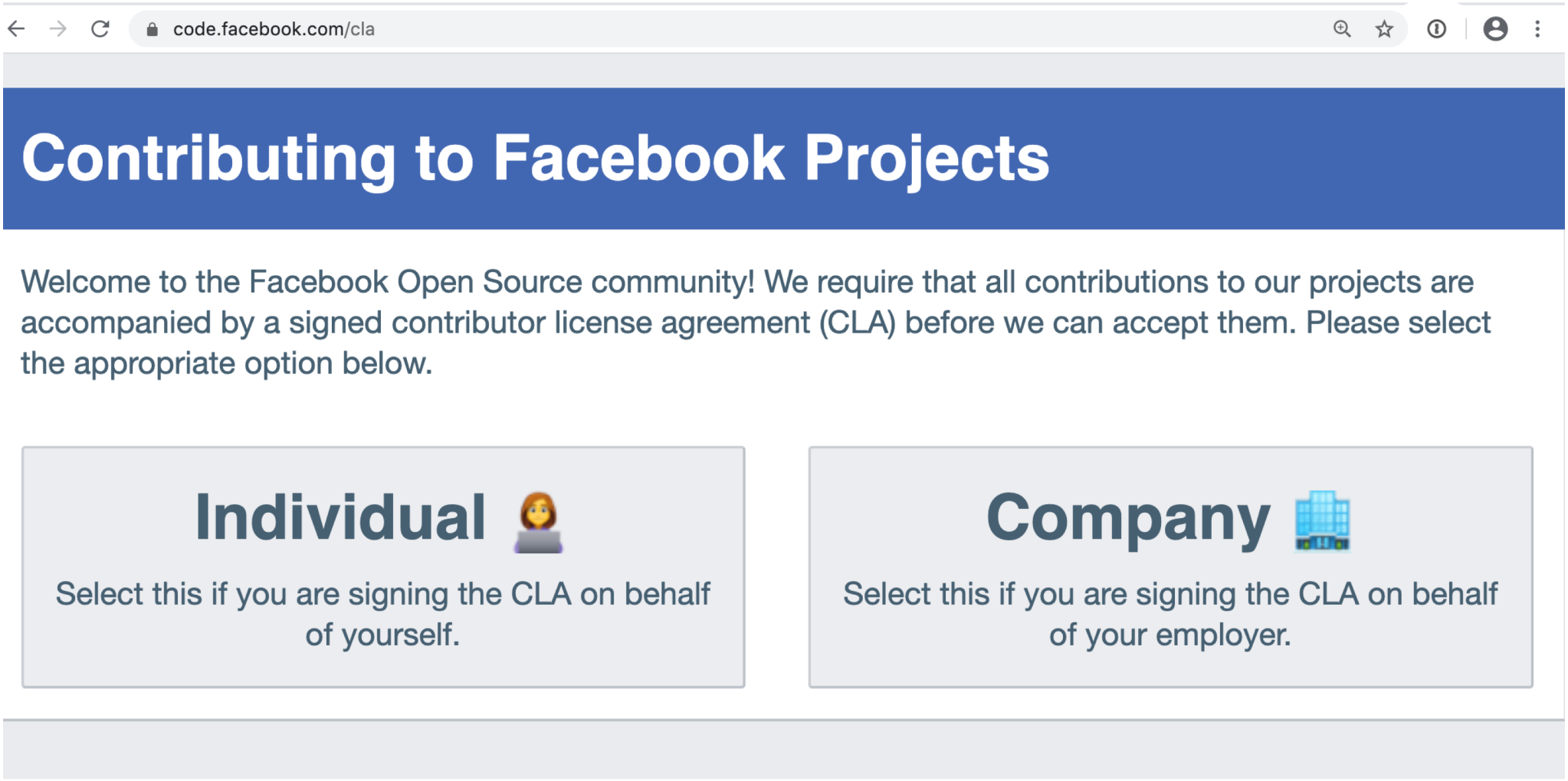
Contributing to Docker

Want to hack on Docker? Awesome! We have a contributor's guide that explains setting up a Docker development environment and the contribution process.

This page contains information about reporting issues as well as some tips and guidelines useful to experienced open source contributors. Finally, make sure you read our community guidelines before you start participating.

Topics

- Reporting Security Issues
- Design and Cleanup Proposals
- Reporting Issues
- Quick Contribution Tips and Guidelines
- Community Guidelines


code.facebook.com/cla
Contributing to Facebook Projects
Welcome to the Facebook Open Source community! We require that all contributions to our projects are accompanied by a signed contributor license agreement (CLA) before we can accept them. Please select the appropriate option below.
Individual
Select this if you are signing the CLA on behalf of yourself.
Company
Select this if you are signing the CLA on behalf of your employer.

Closing Day: April 3

- Poster session
- Poster “competition”
  - Teams will rank / evaluate three other posters
  - Awards for overall top 3
- Drinks

- [Details will follow]

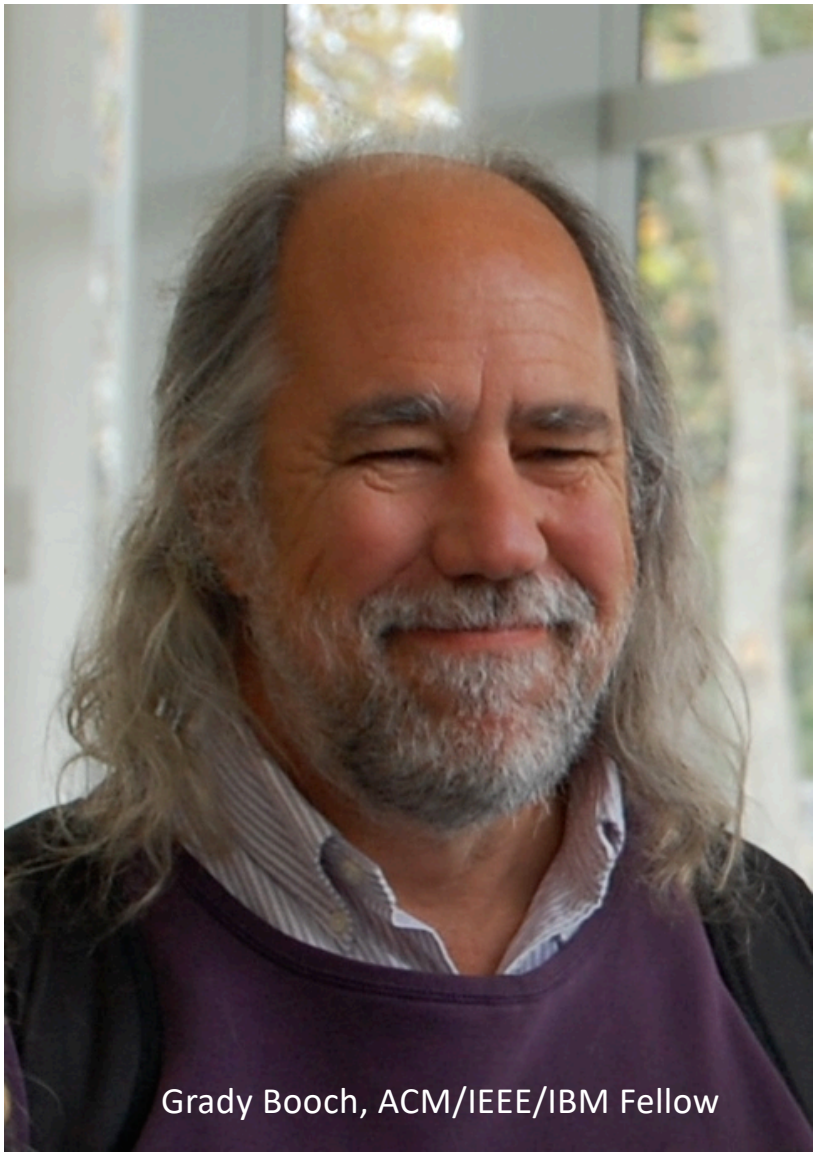
The good software architect is
hungry for new knowledge

What would you ask if you could talk
to a top-architect from IBM?

Architecture Defined

Architecture represents
the significant design decisions
that shape a system
where significant is measured
by cost of change

(Grady Booch, March 2006)



Grady Booch, ACM/IEEE/IBM Fellow

Grady Booch ✓
@Grady_Booch

In truth, only 873 software issues in (approximately) eight million lines of (mostly hard real time) code on board the F35 is an impressivly small number.

That being said, a plane that can't shoot straight is just a very expensive chunk of metal.

F-35's Gun That Can't Shoot Straight Adds to Its Roster of Flaws
Add a gun that can't shoot straight to the problems that dog Lockheed Martin Corp.'s $428 billion F-35 program, including more than 800 software flaws.
bloomberg.com

8:18 PM · Jan 31, 2020 · Twitter Web App

Waldo Jaquith @waldojaquith · Feb 5

When government pays companies to build big custom software programs for them, they succeed just 13% of the time. Now I will tell you why failure is so common, and about the simple change that turns those outcomes on their head.

Show this thread

1:58 AM · Feb 7, 2020 · Twitter for iPhone

10 Retweets **70** Likes

Grady Booch
@Grady_Booch

13%.

That sounds high.

> 
> **Waldo Jaquith** @waldojaquith · Feb 5
>
> When government pays companies to build big custom software programs for them, they succeed just 13% of the time. Now I will tell you why failure is so common, and about the simple change that turns those outcomes on their head.
>
> Show this thread

1:58 AM · Feb 7, 2020 · Twitter for iPhone

10 Retweets **70** Likes

Grady Booch ✓
@Grady_Booch

When you screw up as thoroughly and as publicly as did Shadow on what should have been a relatively straightforward piece of software, you know there is something horrible, fundamentally wrong with your development process.

And the company management thereof.

> **VentureBeat** ✓ @VentureBeat · Feb 4
> Mysterious startup Shadow under scrutiny after Iowa Caucus meltdown wp.me/p8wLEc-aOg5 by @obrien

6:24 PM · Feb 4, 2020 · Twitter for iPhone

Grady Booch

@Grady_Booch

Every line of code has a moral and ethical implication.

Architecting the Unknown

... where science moves from the unknown to the known,
we in computing are largely moving
from the known to the unknown

and in many ways we are constrained
by [...] our own imagination

Grady Booch – SATURN 2016 keynote

Preparation (I): *Watch before February 14*

- SATURN 2016 keynote: *Architecting the Unknown*
  https://www.youtube.com/watch?v=RJ3v5cSNcB8

- ICSE SEIP 2015 keynote: *The Future of Software Engineering*
  https://www.youtube.com/watch?v=h1TGJJ-F-fE

- TED 2016: *Don't Fear Superintelligent AI*
  https://www.youtube.com/watch?v=z0HsPBKfhoI

Preparation II

- *Think of questions you'd like to ask.*
- Grady expects you to be curious and critical
- *Share on Mattermost #ama channel.*
- During AMA, you'll get opportunity to ask them yourself

Delft SWA
@DelftSWA

.@freire_da_silva You do not ask your boss: Can I go to the toilet? You don't just shit in your pants, so do also not just shit on your code

9:45 PM · Mar 2, 2017 · Twitter Web Client