UNIVERSITÄT MANNHEIM

Seizing Continuous Integration process to improve Software Reuse

Master Thesis

submitted: February 2017

by: Piero Antonio Divasto Martínez
born March 10th 1986
in Iquique
Chile

Student ID Number: 1472793

Abstract

On the one hand we have the Douglas McIlroy's vision [39] of a software system composed of already existent components, where you basically put different components, that already exist, together in order to form the system is being built. Although the IT industry has tried for many years to improve the speed and reduce cots of software development by reusing components, this vision is still the exception rather than the rule. On the other hand we have Continuous Integration which is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily, thus leading to multiple integrations per day. Each integration is verified by an automates build (including test) to detect integration errors as quickly as possible. In this work we present a novel approach that improve the reuse of software by seizing continuous integration process. Also we present an unobtrusive tool as a proof-of-concept that extracts interface signatures and test classes from a given project, which are sent to a code-search engine to retrieve similar components based on text-driven and test-driven search techniques. Finally the result is sent by email to the user.

Contents

List of Figures

List of Tables

List of Abbreviations

$ATDD$ .... Acceptance TDD

BDD .... Behavior Driven Development

CC .... Creative Commons

CD .... Continuous Delivery

CI .... Continuous Integration

CVS .... Concurrent Versions System

JIT .... Just in time

SCM .... Source Control Management

TDD .... Test-driven Development

TFD .... Test-first Development

VCP .... Version Control Platform

VCS .... Version Control System

1. Introduction

Building software using already existent components is not new. During the 1960s Douglas MacIlroy came up with the vision of a software system composed of components, where one basically puts different pieces (components), that already exist, together in order to form the system that is being built. During the next 50 years of MacIlroy's vision, the IT industry has tried to improve the speed and reduce the cost of development by reusing software, however this vision is still far for being a reality.

This process has attracted the attention of the industry due to its alleged benefits. Reduction of redundancies, costs reduction, quality improvements, and fostering innovation are some of the benefits that come with reusing software. However this is not an easy task.

The adoption of suitable reuse strategies is pretty challenging as it takes place in a multifaceted environments which incorporate aspects spanning from technical to organizational at different level of abstractions [2]. In general the challenges can be divided in two gorups: (1) organizational and (2) technical [24, 2]. In the former, problems such as organizational structure, inertia, knowledge, management, among others might affect the success of the implementation of this practice. For the latter, four core problems have been identified in the literature to implement a sustainable reuse repository: the repository problem [51], the representation problem [45], the usability problem[18], and the retrieval problem [47].

The rise of open-source movement, the improvements in the Internet connectivity, the advances in database, and the emergence of code-search engines have clearly solved the first problem. Techniques such as ranking approaches like SOCORA [30] or the ranking based on use relations presented by Inoue et al., and the idea of parsing source code in order to extract objects introduced by Koders.com [1] have addressed the representation problem. Furthermore test-driven search technique

---

[1]Discontinued.

[23, 24] promises to solve the representation, usability and retrieval problems, by finding functionally sufficient components based on tests.

On the other hand we have Continuous Integration (CI), which is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily, thus leading to multiple integrations per day. Each integration is verified by an automates build (including test) to detect integration errors as quickly as possible. Many teams has found that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software quicker [11].

Here is when the following questions came up: *How can Software Reuse and Continuous Integration be best married?.*

Software projects that adopt continuous integration normally adopt the agile development technique called test-driven development (TDD), where a test case is created before the actual code that need to be implemented. Thus, in addition with the frequent commits made by the developers, we found a great chance to improve the chances of reusing software components. Every time new code is pushed to the version control server (VCS), the CI server is triggered. At this moment we are able to analyze the source code and extract test classes, along with the interface signature, to sent them to a code-search engine. This engine will retrieve the result candidates that meet the functional requirements of the test classes, and then the result will be sent to the developers. Consequently the team will realize that what they are working on might be already available in the internet so they would not need to re-implement the same component from scratch.

In the reminder of this document we explain with further details how this approach works. First, in Chapter 2 we provide the foundations and concepts needed to understand the approach proposed. There we will start explaining about what software reuse practice means. Then we move to the agile practice test-driven development. After that we explain about continuous integration process. To conclude this chapter, we talk about microservices. In the next Chapter we describe the usage scenario in order to motivate and strengthen the approach which is presented in Chapter 4. There we detail our approach, its design, how we implemented it, the results of applying it and the technologies we used. To close this thesis, in Chapter 5, 6, and 7 we discuss about potential pros/cons of this

work, future work on the idea, and conclusion respectively.

2. Foundations

In this chapter we describe the foundations and concepts necessary to understand the work proposed in this thesis. We begin with describing Software Reuse, its state-of-art, its benefits, challenges, code-search engines, component retrieval techniques, and ranking components. Then we explain about Continuous Integration, its benefits, challenges, and successful CI practices. After that we detail the agile development technique test-driven development (TDD). To conclude with this chapter we describe Microservices architectural style and its characteristics.

2.1. Software Reuse

Douglas McIlroy [39] envisaged in the 1960s a software system composed of already existent components. Where you basically put different components, that already exist, together in order to form the system is being built. One can depict this vision using LEGO where the system is a group of pieces put together, and each of those pieces is a component.

Although the IT industry has tried for many years to improve the speed and reduce costs of software development by reusing components, McIlroy's vision is still the exception rather than the rule.

Despite the fact that it is possible to find several definitions of software reuse in the literature, most of them are similar to the definition proposed by Krueger [32]:

"Software reuse is the process of creating software systems from existing software rather than building software systems from scratch"

Since we already have a formal definition of software reuse, the next question would be: What can be reused?. To this regard, the work of Frakes and Terry defined a list (Table 2.1) of potentially reusable software artifacts, where you can find architectures, source code, requirements, etc.

1. architecture	6. estimates (templates)
2. source code	7. human interfaces
3. data	8. plans
4. designs	9. requirements
5. documentation	10. test cases

Table 2.1.: Potentially reusable aspects of software projects according to [16].

However, reuse traditionally means the reuse of code fragments and components [41]. When we talk about components, we mean any cohesive and compact unit of software functionality with a well defined interface [26]. Therefore a component can be a simple class or even a web-service.

Software reuse emerges as a solution for the so called software development crisis [31]. Organizations face several problems in software development including increased costs, delayed schedules, unsatisfied requirements, and software professional shortage. Therefore, by reusing software components projects can reduce time-to-market, lower development costs, and increase software quality [15].

2.1.1. Benefits of Software Reuse

Software reuse has not failed to attract the attention of industry due to its alleged benefits. In the industry the need of reduction of redundancies, as well as costs reduction and quality improvements is perceived. Moreover, the vision of fostering innovation and market penetration due to shorter production cycles promised obvious strategic business advantages [2]. Research literature has reported benefits from successful adoption of software reuse:

- Lower cost and faster development

- Higher quality

- Standardized architecture

- Risk reduction

Unfortunately the adoption of suitable reuse strategies is pretty challenging as it takes place in a multifaceted environment and, thus, incorporates aspects ranging from technical to organizational at different level of abstractions [2].

2.1.2. Challenges of Software Reuse

Organizational challenges and human factors have been identified as potential inhibitors to a successful implementation of reuse practices [42]. Business strategy, management commitment, and company culture are organizational factors that might affect software reuse [54]. In the work of Bauer and Vetro'[2] the following organizational obstacles were identified in the literature:

- Organization structure: Factors like competition, overlapping or unclear responsibilities, priority conflicts, and lack of coordination of reuse activities jeopardize the successful of reuse implementation.

- Inertia: Some organizations usually assess their managers and developers based on the success of their isolated projects, which incentives local optimization that impact reuse in a company-wide scale.

- Knowledge: If an organization plans to implement reuse practices, it needs clear position organization-wide and research into current methods and techniques for reuse.

- Management: As implementing reuse practices require changes in governance strategies, it causes additional overhead that tends to be underestimated in the initial planning, which put at risk successful of reuse.

- Economic: Implementation of reusability requires investment and long-term support from management to resolve restrictive resource constraints.

- Disincentives: The quality of the reusable component candidates is one of the strongest disincentives. Moreover the criteria applied to measure developers and managers have an impact on their motivation to take part into reuse.

Beside the organizational challenges and human factors described before there are technical aspects identified as potential obstacles. Find the right third-party software component to be reused based on a well-defined specification is one of the most challenging approaches due to the fact that it requires a clear-cut matching of the potential reuse candidate and the given specification. Despite the fact that there are several search tools available in the market, most of them are still text-based and they neither reflect nor support the need to match the reuse candidates with the syntactic and semantic characteristics of a specification [24]. Hummel

and Janjic [24], based on software retrieval literature, identified fours problems
for implementing a sustainable reuse repository.

- Repository problem [51]: This reuse repository should create and maintain
  a big enough software components collection to provide valuable search.

- Representation problem [45]: The repository should represent and index its
  content in a way that makes it easily accessible.

- Usability problem [18]: The repository should permit characterizing a de-
  sired component with reasonable effort and precision.

- Retrieval problem [47]: The repository should execute the queries with high
  precision to retrieve the desired component.

Although the two first challenges have been address over the last years, the last
two have not been addressed completely. By the rise of open-source movement and
the improvements in the Internet connectivity, software developers have got access
to vast areas of free software, thus the problem of sources of components is not
an issue any more. Furthermore, with the advances in database and text search
technologies code-search engines have made the creation of *Internet-scale* software
repositories wherefore this repository concern can be regarded as solved [24]. For
the second problem, clever ranking approaches such as the ranking proposed by
Inoue et al. [28] which ranks those components higher in the result list of a
search that are more often used than others amongst the indexed files. Moreover,
the idea of parsing source code in order to extract objects and their method
introduced by Koders.com, where techniques that addressed the representation
problem. However, the last two problems remain still in the focus of interest in
the research community.

As an attempt to solve the last three technical challenges, Hummel and Janjic
[24] proposed test-driven reuse. There they found that well formulated test cases
reveal enough syntactical information and semantics of a desired component that
they can be used as a query for software searches effectively[24]. Moreover, test-
driven search guarantees results of very high precision considering that the test is
regarded as the definitive description of what functionality is required[4, 24, 30].

2.1.3. Code-search engines

Code-search engines are the heart of the new generations of reuse support tools. For example, Code Conjurer [26] uses Merobase component search engine, Code-Genie relies on Sourcerer [34] and ParseWeb that works over Google Code Search [1] [56]. Nevertheless there are more code-search engines available in the market:

- **searchcode**[2] is a free source code search engine. Code snippets and open source (free software) repositories are indexed and searchable. Most information is presented in such a way that you should not need to click through, but possible if required.

- **NerdyData**[3] is a search engine for source code. It supports HTML, Javascript and CSS. With this search engine one can retrieve the web pages which are using a specific file. For example if one looks up for the CSS file *font-awesome.min.css* it will retrieve the sites that are using this file.

- **Spars-J**[4] is a keyword and name matching code search engine on open source XML, Java and JSPs based on component rank and keyword rank algorithm.

- **ComponentSource**[5] It is a keyword-matching of component descriptions in a marketplace. Here you can find components for different platforms and technologies.

- **Satsy** is a source code search engine that implemented semantic search using input/output queries on mutil-path programs with ranking. Based on survey to 30 participants, it retrieved more relevant results than Merobase [55].

- **Merobase**[6] is a component search engine that uses Lucene [7] to index programming language units from various open source repositories (such as Sourceforge, Google Code, or the Apache projects) and the open Web.

---

[1] Shutdown January, 15 2012 [21]

[2] https://searchcode.com/ (accessed: 13.01.2017)

[3] https://nerdydata.com (accessed: 13.01.2017)

[4] http://sel.ist.osaka-u.ac.jp/SPARS/ (accessed: 13.01.2017)

[5] https://www.componentsource.com/ (accessed: 13.01.2017)

[6] http://www.merobase.com/ - Official project deprecated, we use a local implementation

[7] Apache LuceneTM is a high-performance, full-featured text search engine library written entirely in Java. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform [10].

Merobase, when crawling the code, its analysis software identifies the basic abstraction implemented by a module and stores it in a language agnostic description format. This description includes the abstraction's name, methods names, parameter signatures, among others.

Although Google or Yahoo! are not specialized code search engines, they are able to return more precise results than code search engines in some cases[25].

2.1.3.1. Merobase

Merobase contains special parsers for each supported programming languages (currently it supports Java, C++, and C#, also it supports WSDL files, binary Java classes from JARs and .NET binaries), which extracts syntactical information, stores it in the index and search for it later. Moreover, it contains a special parser for JUnit which is able to extract the interface of the class under test from test cases.

Every time a user make a request to Merobase, the parsers are invoked and try to extract as much syntactic information from the query as possible. If none of the parsers recognize parsable code in the query, it executes a simple keyword search. Based on parsed syntactic information, Merobase supports retrieval by class and operation names, signature matching and by matching the full interface of classes described before.

When a JUnit test case is submitted, a test-driven search is triggered. In this case, Merobase automatically tries to compile, adapt and test the highest ranked candidates. If a candidate is relying on additional classes, the algorithm uses dependency information to locate them as well. It is important to point out that the actual compilation and testing are not carried out on the search server itself, but on dedicated virtual machines within sandboxes. This is due to the fact that this ensures that the executed code does not have the possibility to do anything harmful to the user's system or bring the whole testing-environment down [24].

The current implementation of Merobase, which we used in our prototype, has been enhanced in several ways [30]. Many new metrics are measured on the software components, both statically and dynamically, when the test cases executed on them. Also, the semantic retrieval of components has been improved by adding a better parser for Java components which supports the creation of entire class

hierarchies. With this data, information about the components' class hierarchies can be retrieved and stored.

On the other hand, the current implementation is populated with software components extracted from Maven projects mainly extracted from the Maven Central Repository[8].

2.1.4. Component retrieval techniques

As we stated in the Subsection 2.1.2, several problems of software reuse have been already addressed. Now one of the biggest problems is how to choose the so-called *best component*. To do so, several component retrieval techniques have been proposed. Mili et al. [40] divided the component retrieval techniques in six independent groups:

- **Information retrieval methods** are basically the methods of information retrieval used for retrieving candidates by applying textual analysis to software assets. This group has high recall and medium precision.

- **Descriptive methods** add additional textual description of assets such as keyword or facet definition, to the textual analysis. These are high precision and high recall.

- **Operational semantic methods** use sampling of assets to retrieve candidates. They have very high precision and high recall.

- **Denotational semantic methods** use signature methods for retrieving candidates. The have very high precision and high recall.

- **Structural methods** deal with the structure of the components to retrieve candidates. This group has very high precision and recall.

- **Topological methods** are an approach to minimize the distance between query and reusable candidates. It is difficult to estimate or define recall and precision for these methods [40]. As they rely on a *measurable* retrieval techniques, they can be considered as an approach for ranking the candidate components of a query rather than a retrieval technique [25].

---

[8]https://repo1.maven.org/maven2/ (accessed: 15.01.2017)

In the work of Hummel et al. [25] several retrieval techniques were compared. They compared signature matching, text-based, name-based and interface-driven. These techniques were tested using Merobase search engine. As a result of this comparison the best technique in terms of precision is interface-driven. Another technique that was presented in [23], is test-driven search. This technique uses test cases as a vehicle to retrieve the component candidates that fit better the requirements of the user. It promises to improve the precision of the searches in comparison with other techniques explained before.

2.1.4.1. Test-driven search

Simple techniques for component retrieval such as keyword-driven search or signature-driven search may lead to a tons of candidates from which just a small group might be interesting for the user, in spite of the results that fulfill the search criteria. This is due to the fact that not all of them fulfill the functional properties of the desire artifact. Here is were test-driven search comes to light.

Test-driven reuse approach was first introduced in [23]. It is a technique that emerges as a solution to the problem of retrieving numerous irrelevant component candidates due to large amount of component in a repository. Specifically this technique deals with the usability and retrieval problems we explained before. It relies on test cases written by the developer (step a), then it extracts the different interfaces defined in the test case (step b), then it makes the search of reuse candidate (step c). Then it run the tests and *cleans* the result set of candidates that do not pass the tests (step d and e). Finally it shows the cleaned result set to the user (step f). This is the test-driven reuse cycle which can be seen in the figure 2.1.

An implementation of this *cycle* is Code Conjurer which can be found in [26] and[24]. This implementation is an Eclipse plugin that delivers proactively reuse recomendations by silently monitoring the user's work and triggering searches automatically whenever this seems reasonable. This tool supports interface-base searches and test-driven searches. For the latter, the tool has a background agent that monitors the required interface of the JUnit test case (also called, *class under test* (CUT)) written by the developer. When a change in the CUT happens, Code Conjurer sends the JUnit test to Merobase where the interfaces
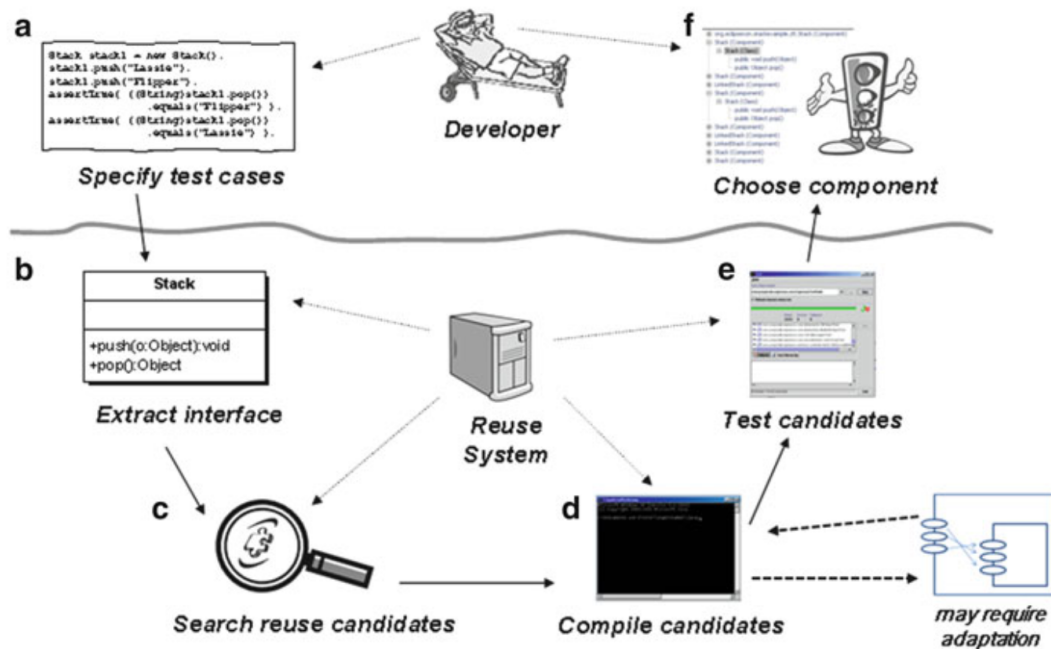
Figure 2.1.: The test-driven reuse *cycle* extracted from [24]

are extracted and used to search for results. Then the candidates retrieved are tested against the test provided and the candidates that do not pass the tests are removed. Finally the candidates are shown to the user in Eclipse. Another tools that uses test-driven search are S6 [48] and CodeGenie [34].

This approach for searching candidates is very promising because of popularity of Agile practices and specifically of Test-Driven development (TDD)[4]. In TDD developers writes the tests of the desired component before writing a single line of its implementation. Therefore, the chances of finding a component that can be reused instead of writing it from scratch is pretty high. We will explain TDD with further details is Section 2.2.

In spite of the fact that this search method retrieves more relevant resources, it needs improvements in order to be applicable for the daily work of a developer. The amount of time taken from submitting the test case until receive the results is sometimes excessive, this is due to the fact that each candidate needs to be compiled and validated against the test case. Although this problem can be overcome by increasing the resources, it brings extra costs.

2.1.5. Ranking components

Retrieving components is not enough. The problem that emerge with retrieving components using code-search engines is when several candidates meet the functional requirements. Which of them is the best?. Normally engines sort the results by keyword or text matching thus the first component in the list is not necessary better than the last one. This is because factors such as performance efficiency, maintainability, and performance efficiency are not being considered for ranking the components.

Several ranking component approaches have been proposed. Inoue et al. [28] presented *ComponentRank* which is a variation of Google's PageRank algorithm that has been adapted to software components. Therefore it ranks components based on use-relations, thus as more often a component is used more relevant it becomes. Satsy[55] presents a new semantic ranking algorithm that basically computes the degree of match between the code query and retrieved code. Reiss' S6[48] combines a semantic code search for components assembled from AST[9] with the ability to rank them in a single-criterion approach using size-based metrics measured using static code analysis. Kessel and Atkinson [30] presented SOCORA which is a component ranking method based on non-dominated sorting that ranks candidates using multiple, non-functional (i.e. quality, performance) criteria indicated by software metrics based only on information about their relative importance[30].

2.1.5.1. SOCORA

The ranking component approach presented in [30] works as follow. First, it establishes a partial ordering of candidates using non-dominated sets determined by non-dominated sorting without assuming any preferences of the user. Depending on the priorities assigned to each selected criterion, it then subrank each non-dominated set in a recursive way until all unique priorities and their corresponding subcriteria are applied to the current level of non-dominated sets, eventually resulting in nested subrankings. Note that in each step, when (new)non-dominated sets are determined by non-dominated sorting, the ranks of sets may change according to their partial ordering. In general, partial subranking are deliberately

---
[9]Abstract Syntax Tree
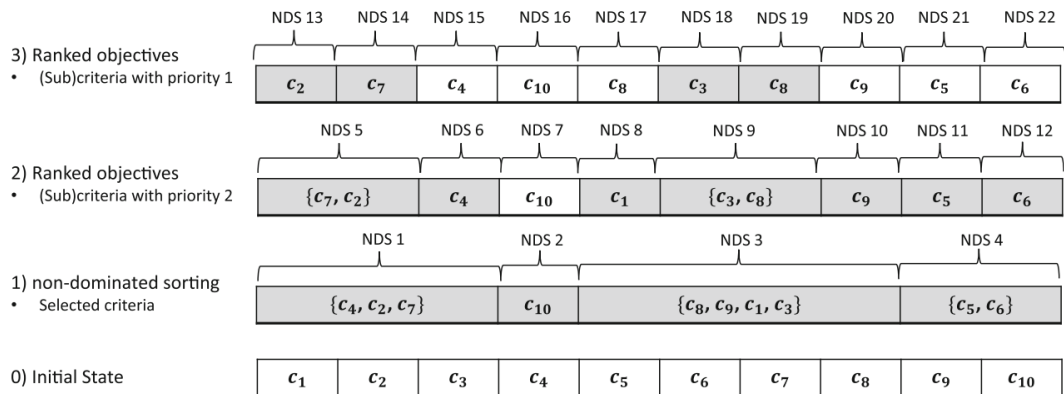
Figure 2.2.: Schematic illustration of SOCORA ranking approach [30]

supported since the user is not required to assign specific priorities to all his/her selected criteria. The priorities are defined by simple integer values (highest value represent more priority than a lower value) with a default priority assignment of 1 for each selected criterion. If the user wants to assign a higher or lower priority to a selected criterion, he/she may increase or decrease the priority by 1 or even a larger number. Both unique and equal priority values may be assigned to selected criteria to establish either a partial or strict ordering. For each priority value, all corresponding criteria are selected to subrank each non-dominated set of non-distinguishable candidates. This process starts with the highest priority value and continues in descending order until the smallest priority value is reached [30]. The figure 2.2 is a schematic illustration of how the component ranking works.

Merobase integration   The prototype of the ranking component approach is integrated with the component search engine Merobase to retrieve the candidates. The figure 2.3 shows how SOCORA[10] is integrated with Merobase.

In the first step the user's functional requirements are described as a JUnit test specification using a HTML5 web-based GUI. Here the user can specify a set of non-functional *quality* criteria that he/she thinks to be important for his/her needs, also he/she optionally can provide relative priorities to partially order them. In the second step, the interface signatures in the JUnit test are extracted and a text-based search is performed in order to find suitable components. In the step 3, the result set of component from the second step are filtered by

---

[10]SOCORA prototype http://socora.merobase.com (accessed: 13.01.2017)

applying the JUnit test in turn, and those that do not match the filtering criteria are removed. Also, during the compilation and execution of the test, the metric selected by the user are evaluated. In the step 4, the information from the last step is used by the ranking algorithm to order the remaining components in the result set. In the step 5 the remaining components in the result set are displayed to the user. In the last step, the user can further analyze the component candidates by exploring the effects of different partial ordering (step 6a), explore different criteria (step 6b), and can group them in different way to shrink the result set (step 6c).
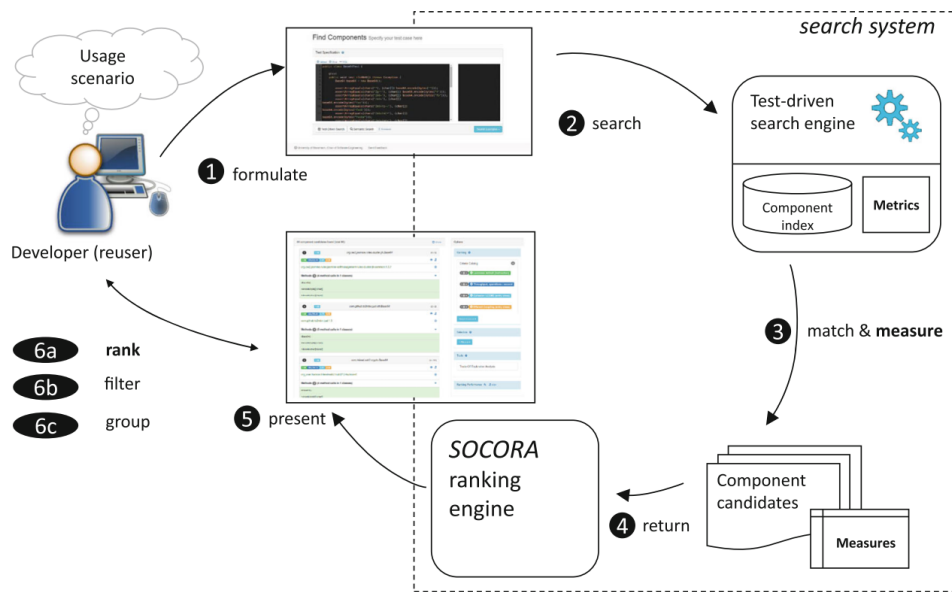

Figure 2.3.: SOCORA and Merobase integration (extracted from [30])

2.2. Test-Driven Development

In this section we will explain Test-driven development (TDD). TDD combines test-first development where you write a test before you write just enough production code to fulfill that test, and refactoring. The primary goal of TDD is to think through your requirements or design before your write your functional code (implying that TDD is both an important agile requirements and agile design technique)[38]. In the reminder of this section we will explain further the definition of TDD, and finally compare it against normal testing.

2.2.1. Definition

The steps of test-first development (TFD) are displayed in the activity diagram of Figure 2.4. The first step is to quickly add a test. Then it is executed, often the complete test suite although one may decide to run only a subset for the sake of speed, to ensure that the new test fails. After that the developer updates the functional code to make it pass the tests. The fourth step is to run your tests again. If they fail he/she needs to update the functional code and retest. Once the tests pass, the next step is to start over. One may first need to refactor the code if needed, in that case we are turning TFD into TDD.

TDD completely inverts traditional development. When a developer first goes to implement a new feature, the first question that he/she asks is whether the existing design is the best design possible that enables him/her to implement that functionality. If so, he/she proceeds via a TFD approach. If not, he/she refactors it locally to change the portion of the design affected by the new feature, enabling him/her to add that feature as easy as possible. As a result he/she will always be improving the quality of the design, thereby making it easier to work with in the future. Figure 2.5 describes the TDD lifecycle just described.

Instead of writing functional code first and then writing testing code, if it is written at all, a developer instead writes test code before the functional code. Moreover, he/she does so in very small steps (i.e. one test and a small bit of proportional functional code at a time). A programmer taking a TDD approach refuses to write a new function until there is a test first that fails because that function is not present yet. In fact, they refuse to add even a single line of

code until a test exists for it. Once the test is in place then they do the work required to ensure that the test suite now passes (i.e. new code may break several existing tests as well as the new one). Although it sounds simple, one needs a great discipline because it easy to forget it and write functional code without first writing a new test.
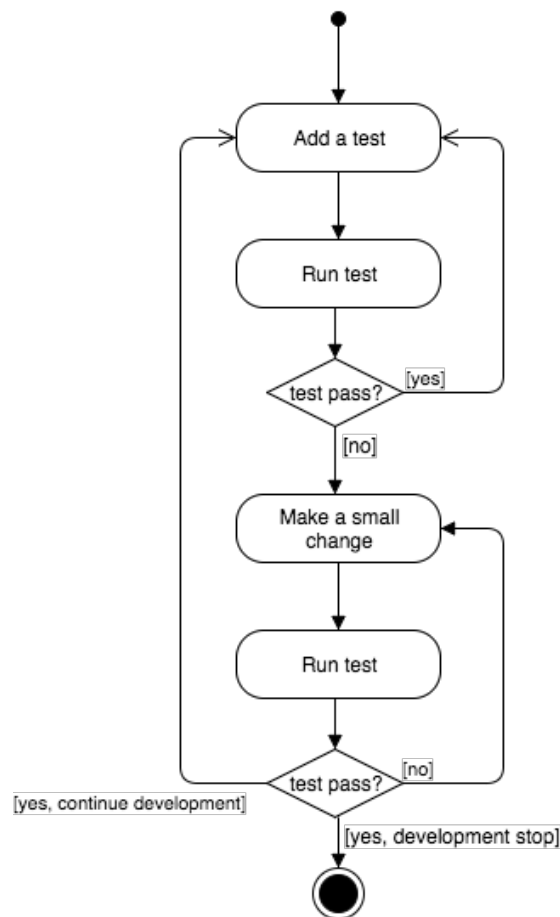
Figure 2.4.: Test-First Development

There are two levels of TDD[1]:

Acceptance TDD (ATDD) is when you write a single acceptance test, and then just enough production functionality code to fulfill that test. The goal of ATDD is to specify detailed, executable requirements for your solution on a just in time (JIT) basis[1]. ATDD is also known as Behavior Driven Development (BDD).
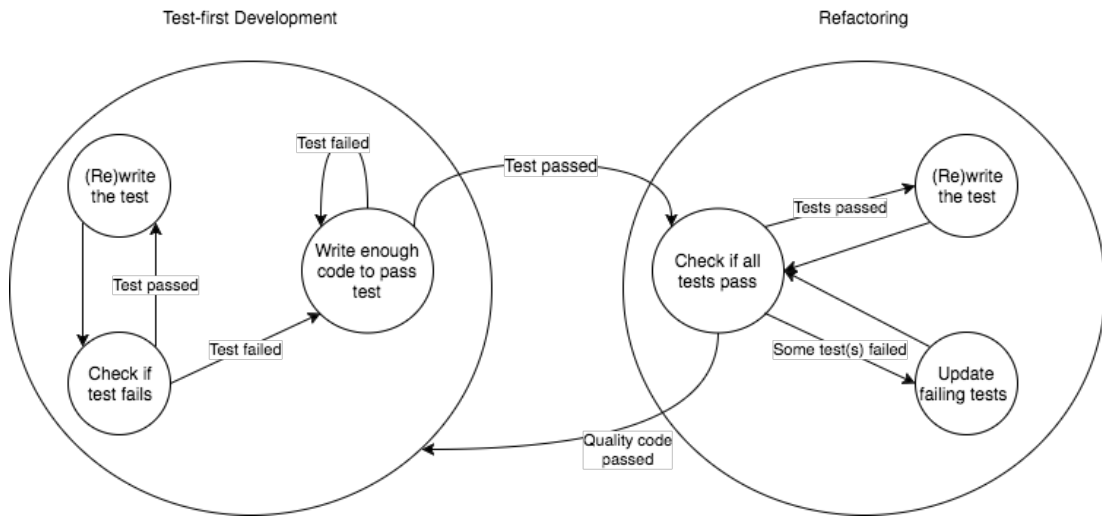
Figure 2.5.: Test-driven Development lifecycle[1]

An acceptance test represents the user's point of view or the external view of the system. It inspects the externally visible effects such as specifying the correct output of a system given a particular input. For instance, an acceptance test can display how the state of an *Order* changes from *paid* to *shipped*. Moreover, it can specify the interaction with interfaces of other systems. Furthermore, acceptance tests can suggest that operations normally hidden from direct testing (such as business rules) should be exposed for testing[29].

These tests are written by the customer, tester, and developer (also called *three amigos*[8]). The customer should create at least one test working with the developer and the tester. The last two then can create more tests and have them reviewed by the customer. The developer connects the tests to the system by writting short bits of code. The idea is that anyone can run the tests, manually or automatically. These tests are written in business domain terms which form a common language that is shared between the *three amigos*[29].

Developer TDD where you write a single developer test, sometimes called unit test, and then just enough production code to fulfill that test. The goal of developer TDD is to specify a detailed, executable design for your solution on a JIT basis. Developer TDD is often simply called TDD[1].
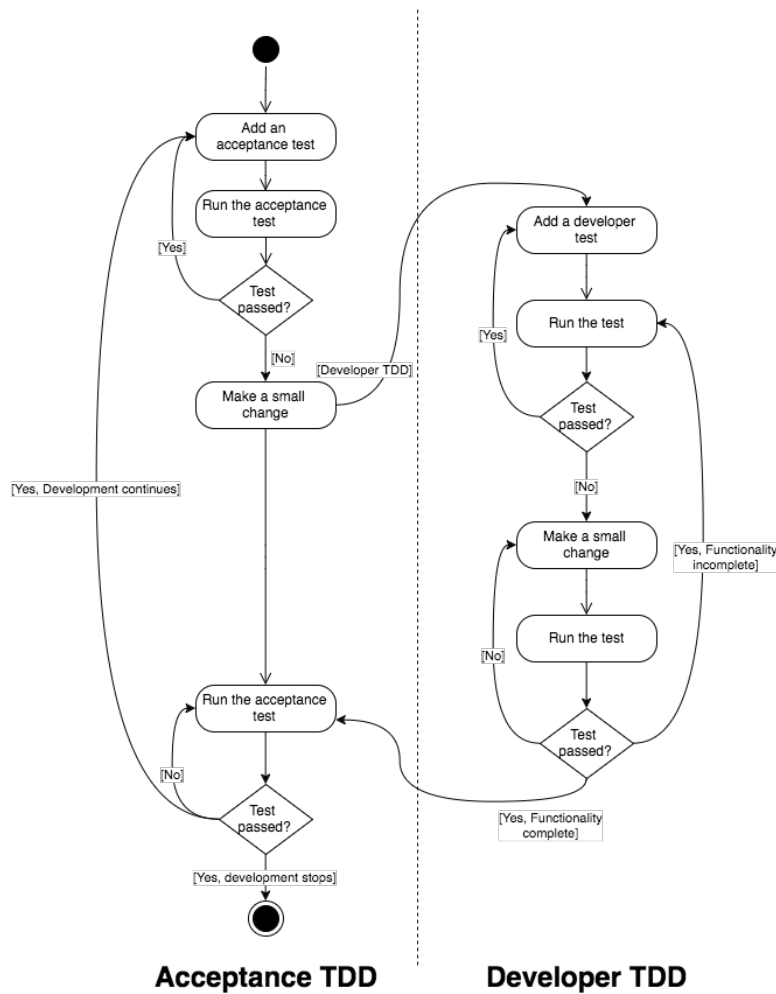
Figure 2.6.: Aceptance TDD and Developer TDD[1]

Figure 2.6 depicts how ATDD and developer TDD fit together. Preferably, it will write a single acceptance test written by the customer, developer and tester. Then run the test and if it passes, add another acceptance test, otherwise make a small change in the acceptance test and add a developer test. After adding the developer test, run it. If it passes, add another developer test, otherwise add enough code to make the test to pass. The developer test cycle ends when the functionality defined by the acceptance test is fulfilled. Then run the acceptance tests to check if all of them passes. The ATDD cycle will end when the development stops otherwise it continues.

2.2.2. Why TDD?

TDD is a practice that has been promoted because it is far more productive than coding in larger steps[1]. Imagine a developer adds a functional code, compile, and test it. It is very likely that the test will be broken by defects that exists in the new code. Therefore, it is easier to find, and then fix, those defects if he/she has written two lines of code than hundreds. Therefore, the most important benefit acquired by applying TDD, is that it enables you to take small steps when developing a software.

TDD facilitates easy maintenance and helps alleviate scope creep. In TDD, code is never written without first writing a test. This results in a great test coverage. Further, the refactoring process ensures written code is as economical as possible, streamlining the code. If there is not a use-case for a piece of functionality, the test is never written, thus the code does not grow.

Test-driven development is a way of managing fear during software development[4]. TDD gives organizations the ability to painlessly update their software to address new business requirements or other unforeseen variables. Since testing in TDD is integrated into the development process at the most granular level, it is guaranteed that every standalone piece of logic can be tested, and thus changed confidently. As described above, by programming in small steps, any error can be overcome quickly. Furthermore, At the end of the application development, there exist thousands of test cases. Therefore, when a software has a considerable complexity, normally making a change in the code is a critical moment for a developer because he/she is scared of breaking something in another part of the application. However, if the project applies TDD from the beginning, when a change is made to the application, all that must be done is run the existing test cases to see if the change has adversely impacted any other piece of the application.

As a conclusion, TDD is a great tool for young companies that seek rapid growth and for older organizations that need to update their legacy systems, and organizations that want to diversify service options or revenue channels.

2.3. Continuous Integration

In this chapter we explain about what Continuous Integration (CI) means, we describe the process steps, its benefits, challenges, and good practices of CI.

2.3.1. What is Continuous Integration?

The term can be found in the context of micro-processes development in the work of Booch et al.. Then it was adopted by Kent Beck in his definition of Extreme Programming [3]. However, it was Martin Fowler who was credited with establishing the current definitions of this practice. Fowler defines CI as following:

Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily, thus leading to multiple integrations per day. Each integration is verified by an automates build (including test) to detect integration errors as quickly as possible. Many teams has found that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapid[11].

CI emerges as a solution for the painful moment of software integration. Despite the fact that the process of integrating software is not a problem for a one-person project, when it increases in complexity it becomes more problematic. For instance, in the old days, a software was divided in modules and each of them were developed independently, once they done, those modules were put together in one step at the end of the project. Doing that led to all sorts of software quality problems, which are costly and often lead to project delays [9].

Figure 2.7 describes the emotional cycle of a manual delivery process[19] when the module integration is necessary. This step is tense moment as errors and failures appeared and they were difficult to find and fix at that stage of the development. In this manner, instead of waiting till the end of modules development to integrate, CI proposes to integrate frequently, usually one person should integrate at least once a day.
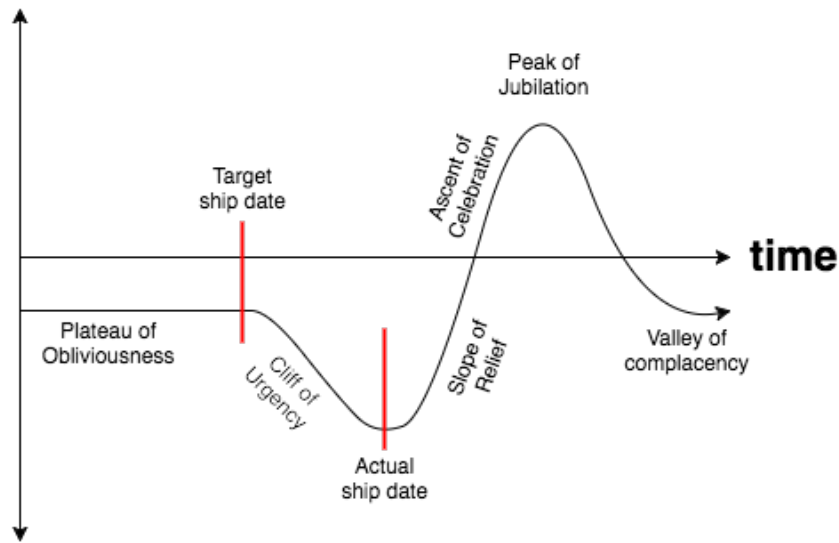
Figure 2.7.: Emotional cycle of manual delivery [19]

Therefore we can separte the CI workflow into the following steps [11]:

CI Workflow

- Developers check out code into their private workspaces

- When done, commit the changes to the repository

- The CI server monitors the repository and checks out changes when they occur.

- The CI server builds the system and runs unit and integration tests.

- The CI server releases deployable artifacts for testing.

- The CI server assigns a build label to the version of the code it just built.

- The CI server informs the team of the outcome of the build.

- In case the build or test failed, the team fixes the issue at the earliest opportunity.

- Continue to frequently integrate and test throughout the project.

This uncomplicated workflow helps the development team to focus on the current code change till it is verified and validated. Moreover, it provides continuous feedback on the quality of the code.

An extension of Continuous Integration in Continuous Delivery (CD). CD is a software development discipline where you build software in such a way that the software can be released to production at any time [13]. Figure 2.8 describes an example of a typical CI/CD pipeline where one can see the relationship between this two disciplines.
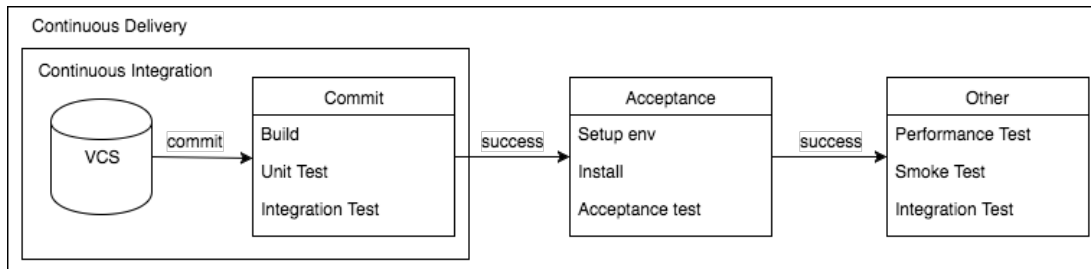
Figure 2.8.: Typical CI/CD pipeline [6]

2.3.1.1. CI Tools

Although the CI is tool-agnostic, the selection of them depends on the project, framework in use, skill-set of the stakeholders and other factors. There are however two must-have tools of any CI system: (1) the Version Control System (VCS) and (2) the CI Server.

The most popular VCSs are SVN, Mercurial, and Git. On top of them, we can find Version Control Platforms (VCP) such as Bitbucket, Gitlab, and Github. In terms of CI servers, we can find Jenkins, Hudson, GoCD as open source projects; TravisCI, CircleCI, CodeShip and Team City as commercial tools.

Therefore choosing the appropriate tools is about finding the balance between price, setup, configuration efforts, ease-of-use, integration capabilities between the selected tools, framework suitability and maintainability in respect to current code base.

Version Control Systems

SVN   Apache Subversion [11] is an open-source software versioning and revision control system, which is used to maintain current and historical versions of files such as source code, web pages, documents, etc. It appeared as an alternative to the Concurrent Versions System (CVS) [44].

One can visualize the SVN filesystem as *two-dimensional*. Two coordinates are used to unambiguously address filesystem items: (1) path, and (2) revision.

Each revision in a Subversion filesystem has its own root, which is used to access contents at that revision. Files are stored as links to the most recent change; thus a repository is rather compact. The system consumes storage space proportional to the number of changes made, not to the number of revisions. Moreover, the Subversion filesystem uses transactions to keep changes atomic. A transaction operates on a specified revision of the filesystem, not necessarily the latest. The transaction has its own root, on which changes are made. It is then either committed and becomes the latest revision, or is aborted. The transaction is actually a long-lived filesystem object; a client does not need to commit or abort a transaction itself, rather it can also begin a transaction, exit, and then can re-open the transaction and continue using it. Potentially, multiple clients can access the same transaction and work together on an atomic change, though no existing clients expose this capability [44].

Mercurial   Mercurial[12] is a free, distributed source control management tool. It offers you the power to efficiently handle projects of any size while using an intuitive interface. It is easy to use and hard to break, making it ideal for anyone working with versioned files.

Mercurial's major design goals include high performance and scalability, decentralized, fully distributed collaborative development, robust handling of both plain text and binary files, and advanced branching and merging capabilities, while remaining conceptually simple.[3] It includes an integrated web-interface. Mercurial has also taken steps to ease the transition for users of other version control systems, particularly Subversion [36].

---

[11]https://subversion.apache.org/ (accessed: 13.01.2017)
[12]https://www.mercurial-scm.org/ (accessed: 13.01.2017)

Git   Within the several different VCSs available, Git is one of the most pop-
ular[13]. Git emerges in 2005 as an alternative to BitKeeper[14] after the break
of the relationship between the commercial company behind BitKeeper and the
community that developed the Linux kernel [6].

The main difference between Git and the other VCSs is the way it thinks about
its data. Other VCSs such as Subversion, CVS, Perforce, and so on, think of the
information they keep as a set of files and the changes made to each file over time.
On the other hand, Git thinks its data like a set of snapshots of a miniature file
system. Therefore, every time a user commits, or save the state of the project,
Git basically takes a picture of what all the files look like at that moment and
store a reference to that snapshot. In addition, Git has three main states where
the files can reside in: committed, modified, and staged. Committed means that
the data is stored in the local database. Modified means that the file was modified
but it has not been committed yet. And staged means that the modified file has
been marked to go into the next commit [6]. The figure 2.9 depicts these three
states.
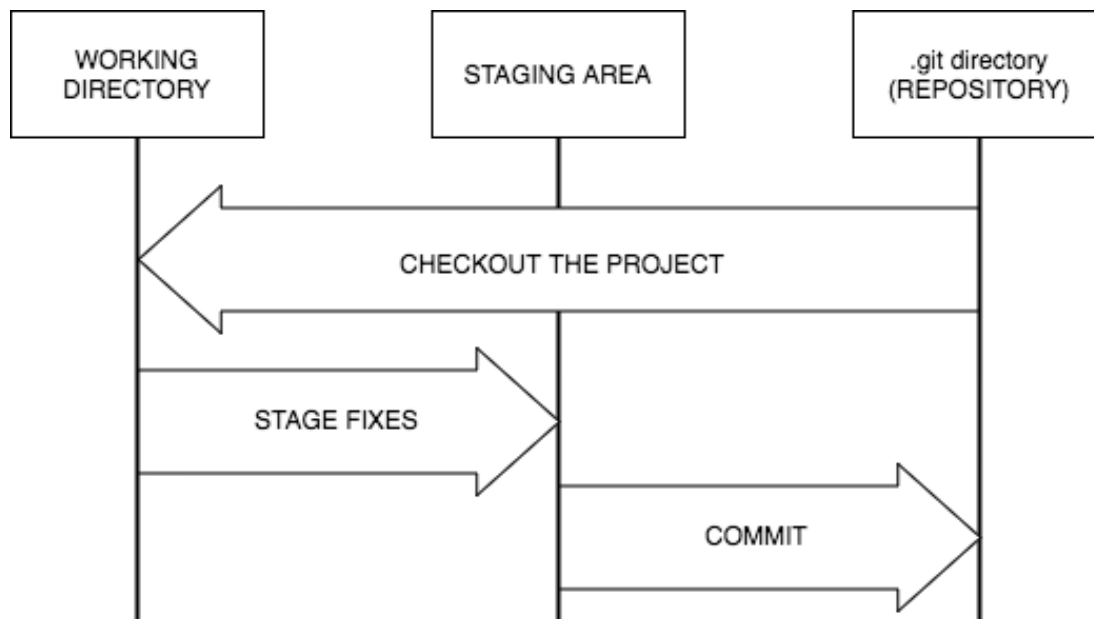
Figure 2.9.: The three main states of Git [6]

The git directory is the core of Git as it is where the metadata and object database

---

for a project is stored. The working directory is a single checkout of one version of the project. Finally, the staging area is a file which is generally stored in the Git directory. This file contains information about what will go into the next commit.

Version Control Platforms

Bitbucket is a web-based hosting service for projects. It supports Mercurial and Git SCMs. Bitbucket[15] offers both commercial plans and free accounts. It offers free accounts with an unlimited number of private repositories. Bitbucket is written in Python using the Django web framework. It started as an start-up by Jesper Nohr, but later it was acquired by Atlassian[16].

Gitlab is an open-source web-based Git repository developed by GitLab Inc. The software was written mainly in Ruby by Dmitriy Zaporozhets and Valery Sizov. Although it offers, in general, the same features like Mercurial and Github, the main characteristics is that it allows you to deploy your own instance of GitLab in your own server.

Github is a web-based Git repository hosting service, which offers all the functionality of Git and its own features. It provides several collaboration tools such as wikis, bug tracking, feature requests, and task management as well as access control. Its development started in 2007 as a side project of P. J. Hyett and Chris Wanstrath [57] and it was officially lunch on the 10th of April in 2008. Since then, Github[17] has gain popularity through the community reaching more than 50 million projects hosted nowadays.

Continuous Integration Servers

Teamcity is a Java-based build management and continuous integration server from JetBrains[18]. TeamCity[19] is commercial software and licensed under a pro-

---

[15]https://bitbucket.org (accessed: 21.02.2017)

[16]https://www.atlassian.com/ (accessed: 21.02.2017)

[17]https://github.com/ (accessed: 13.01.2017)

[18]https://www.jetbrains.com (accessed: 21.02.2017)

[19]https://www.jetbrains.com/teamcity/ (accessed: 21.02.2017)

CI Server	Platform	License	Builders	Integrations	IDE Support	SCM Support		
						SVN	Mercurial	Git
Teamcity	Web Container	Propietary	MSBuild, NAnt, Visual Studio, Ant, Maven 2-3, Gradle, Rake, command-line	Jira, Bugzilla, FindBugs, PMD, dotCover, NCover	Eclipse, Visual Studio, IntelliJ IDEA, WebStorm, PhpStorm, RubyMine, PyCharm	✓	✓	✓
TravisCI	Hosted	MIT	Ant, Maven, Gradle	Github, Heroku	None	✕	✕	✓
Jenkins	Web Container	CC and MIT	MSBuild, NAnt, Ant, Maven, Kundo, Gradle, shell scripts, command-line	Bugzilla, Google Code, Jira, Bitbucket, Redmine, FindBugs, Checkstyle, PMD, Mantis, Trac	Eclipse, IntelliJ IDEA, NetBeans	✓	✓	✓

Table 2.2.: Comparison of Continuous Integration servers

prietary license. It offers several features such as VCS interoperability, cloud integrations, code quality tracking, continuous integration, user management, system maintenance, build history, and extensibility and customization.

TravisCI is a hosted, distributed continuous integration and continuous delivery service used to build and test software projects hosted in GitHub. It offers a free open-source[20] and an enterprise version[21]. Currently more than 300.000 projects use TravisCI (free version).

Jenkins Jenkins is an open source automation server developed in Java. It was originally founded in 2006 as *Hudson*, however a disputed with Oracle the community behind Hudson decided to change the project name to Jenkins[22].

Jenkins enables developers to reliable build, test, and deploy their software. Its extensible and plugin-based architecture has permitted to create a tons of plugins to adapt the CI server to a multitude of build, test, and deployment automation workloads. In 2015, Jenkins surpassed 100.000 known installation making it the most widely deployed automation server[23].

In Table 2.2 we can see a comparison of the three CI Servers described above.

---

[20]https://travis-ci.org/ (accessed: 21.02.2017)
[21]https://enterprise.travis-ci.com/ (accessed: 21.02.2017)
[22]http://archive.is/fl45 (accessed: 13.01.2017)
[23]https://jenkins.io/press/ (accessed: 13.01.2017)

2.3.2. Benefits in adopting CI/CD

Several are the benefits that come with the adoption of Continuous Integration in a software project. In this section we will describe five areas that are improved after CI implementation [49]. It is important to point out that these benefits come along with other practices such as agile transformation [33] and lean software development [46].

- Shorter time-to-market: Many benefits come with the adoption of frequent releases. With fast and frequent releases is possible to get feedback quickly from customer and market, thus the organization gets better understanding of their needs expectations [43]. With that is possible to focus on the most relevant features. Another benefit of delivering frequently is waste reduction as features can be deployed as soon as they are done [35]. Furthermore, with frequent releases is possible to experiment with new features easily and with low impact [43].

- Rapid feedback: With frequent releases is possible to show progress to customers and by that get feedback quickly. With that the development team can focus on important features instead of waste time in features that are not relevant for the customers or the market.

- Improved software quality: Researches have reported a decreased in the number of open bugs and production incidents after adopting CI practices [37] and the link between software quality improvement and the heavy reliance on automated test combined with smaller more manageable releases [35].

- Improved release reliability: In the work of [43] was proved that a working deployment pipeline along with intensive automated testing and fast rollback mechanism possitively affects release reliability and quality. With small and frequent releases fewer things can go wrong in a release according to [13]. In fact, reduction in the stress of developers and other stakeholder have been found by adopting CI [43] [7].

- Improved developer productivity: By automating deployment process, environment configuration and other non-value adding tasks, significant time saving for developers have been observed [50]. Also, in the work of [22] was

observed that although the setup cost of a deployment pipeline can be high, after it is setup developers can focus on value adding software development works.

2.3.3. Challenges in adopting CI/CD

Adopting CI can be very beneficial for a software project as described above, however we can face some challenges that can jeopardize the successful implementation of it. According to the literature, there are seven common challenges in the implementation of a CI:

- Change resistance: Transforming the development of a project towards continuous integration practices requires investment and involvement from the entire organization [50]. Therefore any transformation in how an organization works, will receive a resistance on both a personal level and decision level within the organization.

- External constraint: As a software project is part of a context, external constraint may appear. Normally customer preferences and domain imposed restrictions are sources of constraints. For example in highly restricted domains, legal regulations may require extensive testing before new version can be allowed to enter production [50].

- QA effort: The automated tests suit needs to be exhaustive enough in order to ensure the quality of what is being built. Thus it can lead to increase QA efforts due to difficulties in managing the test automation infrastructure [50].

- Legacy code: Normally legacy software has not been design for being automatically tested and may cause integration failures which may inhibit the continuous deployment process. The ownership of legacy code might belong to another company or team which shift the testing responsibility and this might delay the deployment process.

- Complex software: If the complex if the software project is high, then setting up the CI workflow is more challenging [35].

- Environment management: Keeping all the environments used in development, testing and production sync and similar can be challenging. This is

due to the fact that differences in the environment configuration can lead to undetected issues appear in production. Therefore is essential to have a good configuration management that provisions environments automatically [35].

- Manual testing: Although automated tests are very beneficial, some aspects of software need to be manually tested such as security issues, performance and UX/UI. Therefore heavy manual testing can impact the overall speed and smoothness of the process [35].

2.3.4. Successful CI practices

In order to reduce the risks in adopting CI practices, eight principles, based on several literature reviews, were defined in [49] which should guide the CI implementation within organizations. These principles are generic solutions which can be adapted in most cases.

- Automate the build: The task that triggers the whole pipeline is the commit. After each commit, the next step should be build the binaries. The executable that result of the build, should go through the pipeline until it gets validated, verified and deployment ready.

- Make your build self-testing:

- Every commit should build on an integration machine

- Keep the build fast

- Keep the build green

- Test in a clone of the production environment

- Everyone can see what is happening

- Automate deployment

These 8 principles should guide the CI/CD implementation within organizations. The principles have been stablished as best practices through the validation of these concepts through several literature reviews [[50]; [37]; [53]]. They can be viewed almost as a design pattern for adopting CI/CD, offering a generic solution adaptable to most cases.

2.4. Microservices

Microservices is a software architecture is usually linked to Martin Fowler in its article [14]. It is an architectural styles that aim to develop applications as a suite of small services independently of each other. Each of this services runs in its process and they communicate with lightweight mechanisms (e.g. HTTP resource API). These services are built around business capabilities and independently deployed by fully automated deployment machinery. Also there is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies [14].
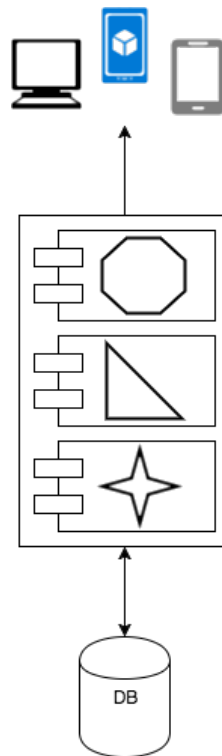
Figure 2.10.: Monolithic architectural style

This architectural style emerges as a solution to the monolithic style due to its problems. An application built using monolithic architectural style is considered as a single unit with normally three main parts (Figure 2.10): a client-side user interface (i.e. HTML, CSS, JavaScript running on user's machine), a database and a server-side application. The latter is a monolith as it handles HTTP request, execute domain logic, retrieve and update data from the database, and populates

and/or select the HTML views sent to the client-side. As it is a single logical
executable, any changes made in the system, no matter the size of it, require a
build and a deploy of the whole server-side application. It is hard to keep a good
modular structure, making it harder to keep changes that must affect one module
within that module. Also it is hard to scale as it is necessary to scale the entire
application rather than parts of it requiring greater resources. These problems are
causing frustrations on people that are using this style, specially nowadays when
applications are deployed to the cloud, leading to the microservices architectural
style.

Microservices architectural style proposes to split the software into independent
services (Figure 2.11). Splitting a software into services brings benefits such as
services as components are independently deployable making it easy for changes,
or services have more explicit component interfaces by using explicit remote call
mechanisms. Moreover, this approach prefers letting each service manage its own
database instead of using a central database for all the services. Applications
developed with this style have meany benefits such as single responsibility, high
scalability, easy to enhance, and ease of deployment. In the following subsection
we will discuss about benefits, challenges and common characteristics that can
be found in applications developed following microservices architectural style.
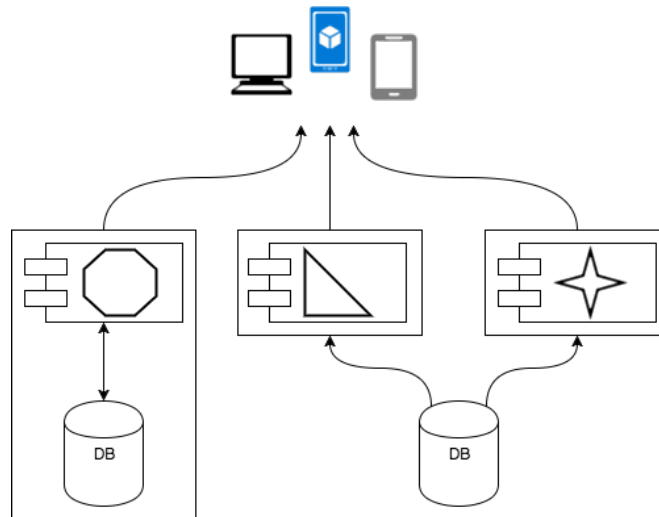
Figure 2.11.: Microservices architectural style

Characteristics of a Microservice Architecture

The work of Fowler and Lewis [14] describes a list of common characteristic of applications built using microservice architectural style. It is important to point out that all the applications using this style must confirm every characteristic of this list.

Componentization via Services

First of all it is important to explain the differences between libraries and services as the primary way of componentizing is by breking down software into services. On the one hand libraries are components that are linked into a application and called using in-memory function calls. On the other hand services are out-of-process components which communicates with a mechanism such as a web service request, or remote procedure call [14].

Splitting a software into services brings benefits such as services as components are independently deployable making it easy for changes, or services have more explicit component interfaces by using explicit remote call mechanisms.

Organized around Business Capabilities

Normally in monolithic applications the development teams are divided by technology layers such as front-end team, server-side team and database team. However when team are divided in this way, even simple changes can lead to a cross-team project taking time and budgetary approval.

In Microservices approach this is taken in a different way by dividing the application into services organized around business capability. Thus services involve all needed for that business area, including user-interface (UI), database, business logic, and any other external collaboration. Therefore each teams are cross-functional, including the full range of skills required for the development of it.

Products not Projects

Usually software development follows a project model where the objective is to deliver some piece of software which, when it is done, it is delivered to a maintenance team and the development team is disbanded.

Microservices approach follows the product model instead of project model. In the product model the development team takes full responsibility of the

software, in other words it owns the product. This results in day-to-day contact of the developers with the users as they have to take on at least some of the support. Moreover this mentality ties in with the linkage to business capabilities as the focus is on how the software can assist its users to enhance the capabilities instead on put the focus on a list of functionality.

Smart endpoints and dumb pipes

Microservices use the approach of smart endpoint and dumb pipes. Instead of putting so much smart into the communication mechanism itself, microservices approach aim to be as decoupled and as cohesive as possible. In other word, a microservice works as a black-box which receives a request, applies a logic, and produces a response. Usually two protocols are used, HTTP request-reponse with resource API's and lightweight messaging (e.g. RabbitMQ, ZeroMQ).

Decentralized Governance

Using a centralized approach brings consequences such as standardise on single technology which keeps the team to use the right tool for the job. In contrast to that microservices approach allows to use use the technology that fit the requirements best. For instance, if NodeJS is more appropriate for a service than Java there will not problem choosing it.

Decentralized Data Management

Microservices approach prefer letting each service manage its own database, either different instances of the same database technology, or entirely different database systems - an approach called Polyglot Persistence [**?**].

Although this approach has some implications for managing updates or increment of complexity, the benefits are worth it. The common approach to solve the former implication is using transactions but implement it in services is difficult, that is why microservice architectures emphasize transactionless coordination between services, assuming that consistency may be only eventual consistency and problems are dealt with by compensation operations. This is useful in order to respond quickly to demand. In spite of the fact that by using different technologies increases the complexity, it allows the teams to solve the problem with the right tool. For instance if a service needs to just retrieve pages by looking up its ID, a key-value database is more suitable than a relational database. Moreover, as many

NoSQL databases are designed to operate over clusters, they can tackle larger volumes of traffic and data which is harder with vertical scaling [12].

Design for failure

As a consequence of using microservice architecture, is that applications need to be designed so that they can tolerate the failure of services. As any service call could fail due to unavailability of the supplier service, the client service must respond as neat as possible.

Although this can be considered as a disadvantage in comparison to monolithic approach, using good monitoring and logging mechanism would help to overcome this. For instance a dashboard which display the status of the different services, current throughput, and latency would help to react in case one or more services go offline.

In conclusion, although microservices architectural style comes with many benefits, it also comes with several difficulties. By splitting up an application into services, we can simplify and speed up the release process, use the right tool for a job, or focus in how to assist better the user instead of a set of functionality. However we have to worry about changes to one service breaking its costumer, or deal with the increment of complexity.

It is important to consider that this style is not a silver bullet. As Fowler described in his blog post [14], the core of The Guardian website [24] was built as a monolith, however every new feature has been added as a microservice that uses the core's API.

---

[24]The Guardian: `https://www.theguardian.com/` (accessed: 04.02.2017).

3. Usage Scenario

Before getting into the approach presented in this work, we will propose a usage scenario to strength and motivate it. The following scenario is based on the usage scenario found in [30].

First of all we will assume that the test software project has the following characteristics. The project has adopted Continuous Integration as we described in Section 2.3, using Git as Version Control System, Jenkins as CI Server, and Java as programming language. Moreover the software is developed following the Test-driven development process.

Envisage the following scenario in which a development team is working or has code ownership of a web service application, and recognizes that she/he requires the ability to encode and decode information in the common Base64 format (e.g., the Base64 variant used in MIME transfer encoding [17]).

As the project is being developed using TDD, developers create a test case that defines a function before it is implemented. For instance, if we follow the scenario, a member of the team will implement the method *encode* which accept an array of bytes decoded in Base64 and it will return an array of chars which represents the input decoded. Moreover the method *decode* that receives an array of chars and returns an array of bytes that represents the input encoded will need to be implemented too. However, as they are using TDD, this member must create a test of these methods before writing functional code. Listing 3.1 represents the methods *encode* and *decode* and Listing 3.2 represents the test class for these methods.

```
package de.unimannheim.informatik.swt.simile.poc.de.unimannheim.
    informatik.swt.simile.poc.base64;


public class Base64 {

    public char[] encode(byte[] decoded) {
        throw new UnsupportedOperationException();
```

```
          }

          public byte [] decode (char [] encoded ) {
10             throw new UnsupportedOperationException ( ) ;
          }

   }
```

Listing 3.1: Base64.java

```
   package de.unimannheim.informatik.swt.simile.poc.base64;

   import de.unimannheim.informatik.swt.simile.poc.de.unimannheim.
       informatik.swt.simile.poc.base64.Base64;
   import org.junit.Test;
5
   import static org.junit.Assert.assertArrayEquals;

   public class Base64Test {

10     @Test
       public void test_rfc4648 () throws Exception{
           Base64 base64 = new Base64 ( ) ;

           assertArrayEquals ( chars ("") , (char []) base64.encode( bytes (""
               ) ) ) ;
15         assertArrayEquals ( chars ("Zg==") , (char []) base64.encode(
               bytes (" f " ) ) ) ;
           assertArrayEquals ( chars ("Zm8=") , (char []) base64.encode(
               bytes (" fo " ) ) ) ;
           assertArrayEquals ( chars ("Zm9v") , (char []) base64.encode(
               bytes (" foo " ) ) ) ;
           assertArrayEquals ( chars ("Zm9vYg==") , (char []) base64.encode(
               bytes (" foob " ) ) ) ;
           assertArrayEquals ( chars ("Zm9vYmE=") , (char []) base64.encode(
               bytes (" fooba " ) ) ) ;
20         assertArrayEquals ( chars ("Zm9vYmFy") , (char []) base64.encode(
               bytes (" foobar " ) ) ) ;

           assertArrayEquals ( bytes ("") , (byte []) base64.decode ( chars (""
               ) ) ) ;
           assertArrayEquals ( bytes (" f ") , (byte []) base64.decode ( chars ("
               Zg==" ) ) ) ;
```

```
            assertArrayEquals(bytes("fo"), (byte[]) base64.decode(chars(
                "Zm8=")));
25          assertArrayEquals(bytes("foo"), (byte[]) base64.decode(chars
                ("Zm9v")));
            assertArrayEquals(bytes("foob"), (byte[]) base64.decode(
                chars("Zm9vYg==")));
            assertArrayEquals(bytes("fooba"), (byte[]) base64.decode(
                chars("Zm9vYmE=")));
            assertArrayEquals(bytes("foobar"), (byte[]) base64.decode(
                chars("Zm9vYmFy")));
        }
30
        protected byte[] bytes(String s) throws Exception {
            return s.getBytes("ASCII");
        }

35      protected char[] chars(String s) throws Exception {
            return s.toCharArray();
        }
    }
```

Listing 3.2: Base64Test.java

After that, this programmer will write enough code to fulfill the test case. This
enough code neither need to be the *best* nor the last version, because the next
step will be to refactor it. Listing 3.3 represents the enough code to pass the test.

```
package de.unimannheim.informatik.swt.simile.poc.de.unimannheim.
    informatik.swt.simile.poc.base64;

public class Base64 {

5       public char[] encode(byte[] decoded) throws
            UnsupportedEncodingException {
          if(Arrays.equals("".getBytes("ASCII"), decoded))
              return "".toCharArray();
          if(Arrays.equals("f".getBytes("ASCII"), decoded))
              return "Zg==".toCharArray();
10        if(Arrays.equals("fo".getBytes("ASCII"), decoded))
              return "Zm8=".toCharArray();
          if(Arrays.equals("foo".getBytes("ASCII"), decoded))
              return "Zm9v".toCharArray();
          if(Arrays.equals("foob".getBytes("ASCII"), decoded))
```

```
15          return "Zm9vYg==".toCharArray();
        if(Arrays.equals("fooba".getBytes("ASCII"), decoded))
            return "Zm9vYmE=".toCharArray();
        if(Arrays.equals("foobar".getBytes("ASCII"), decoded))
            return "Zm9vYmFy".toCharArray();
20      return "".toCharArray();
    }

    public byte[] decode(char[] encoded) throws
        UnsupportedEncodingException {
        if(Arrays.equals("".toCharArray(), encoded))
25          return "".getBytes("ASCII");
        if(Arrays.equals("Zg==".toCharArray(), encoded))
            return "f".getBytes("ASCII");
        if(Arrays.equals("Zm8=".toCharArray(), encoded))
            return "fo".getBytes("ASCII");
30      if(Arrays.equals("Zm9v".toCharArray(), encoded))
            return "foo".getBytes("ASCII");
        if(Arrays.equals("Zm9vYg==".toCharArray(), encoded))
            return "foob".getBytes("ASCII");
        if(Arrays.equals("Zm9vYmE=".toCharArray(), encoded))
35          return "fooba".getBytes("ASCII");
        if(Arrays.equals("Zm9vYmFy".toCharArray(), encoded))
            return "foobar".getBytes("ASCII");
        return "".getBytes("ASCII");
    }
40
}
```

Listing 3.3: Base64.java with implementation to pass the test

Additionally, as the team adopted CI, developers should commit often the changes to the Git server. After adding enough functional code to fulfill the test case, the developer is able to push new code to the Git server. This action triggers the CI server which checkouts the code and runs the unit tests available. Therefore going along with the example given above, the class *Base64.java* and the test class *Base64Test.java* should be committed to the repository.

From this simple scenario emerge the following questions:

How might the team realize that what they are working on has not been developed already?

If we follow the example, the implementation of encode and decode in Base64 is very easy to find, however that does not mean that the developer is aware of that at the moment he/she is developing. Therefore it would be very useful that an automated tool analyzes the code and recommends similar components that implement the functionality being developed.

How might we seize continuous integration in order to improve chances of reusing software?

Continuous integration process seems to be a very good opportunity for software reuse. According to CI workflow explained in Section 2.3, page 22, every change made in the code should be committed and pushed to the VCS. This pushed code is tested by the CI server and accepted if it passes all the tests otherwise it is rejected. This is a good chance to extract method signatures and make the code search named previously.

Would TDD be helpful?

According to TDD process before implementing a method a failure test should be implemented before. If we assume that and that every code change is committed and pushed to the VCS, we will also be able to extract the test classes to search for similar components.

In the following Chapter we will answer these questions and present our approach that improve that chances of reusing software by seizing the use of Continuous Integration.

4. Simile

In this chapter we present our approach that seizes Continuous Integration process in order to recommend similar components to the one that are being developed in a software project. The idea of integrating CI and Software reuse emerged with the following research question: *How can code search engines be best married with CI?*. The objective was to find a way where a team could seize the continuous integration process in order to find similar components to the components they are developing. Thus they would be able to reuse this component instead of implementing them from scratch reducing development time and costs. As an attempt to answer the question we implemented a tool (proof of concept) called Simile. This tool analyses the source code and extract information needed (e.g. interface signature) to find similar components.

Following the usage scenario explained previously in Chapter 3, we will describe how our proposed approach works. In the first step (figure 4.1 - 1) the developer commits code to its local VCS. Then when the developer decides to push changes to the remote repository, the CI server is triggered (figure 4.1 - 2). In the next step (figure 4.1 - 3), Simile will analyse the code and extract information such as interfaces signatures and test classes. This information will be sent to SOCORA (figure 4.1 - 4) in order to find similar components. Then it will return a ranked list of similar components. Simile will receive this information and send this result to the developers by email (figure 4.1 - 5).

The main objective of Simile is to help the team to find similar components so they would be able to reuse them instead of implementing them from the very beginning. Thus they would able to reduce time development and costs through reuse components that are already tested and ready to work.

In the remainder of this Chapter we explain in details how Simile was designed and implemented. Then we detail how it is integrated into Continuous Integration server (Jenkins). After that we will explain how we integrate Simile and SOCORA for searching for similar components.
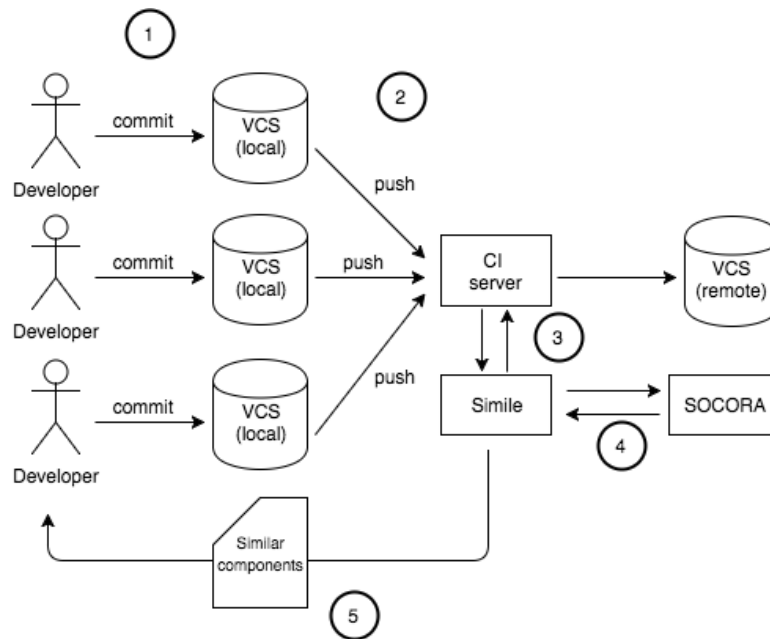
Figure 4.1.: Simile approach

4.1. Design

Simile was formulated as a microservice which was designed taking in account that there are several CI servers available in the market. Although we used Jenkins as CI server for our proof-of-concept, Simile was designed as an agnostic tool. Figure 4.2 depicts an overview of the design of our implementation.
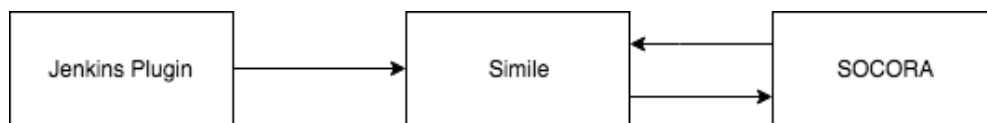
Figure 4.2.: Overview of implementation's design

Despite the fact that the design is rather simple, this allows to connect Simile to any CI server available plugins without changing the core. For instance, if in the future we would like to support another CI servers such as Teamcity, TravisCI, or Buddy, we would just need to develop a simple plugin with the same features of the current plugin developed for this thesis.

Figure 4.3 represents the domain model of Simile. The main class is *Simile*, which

is a service that is in charge of starting the process of analyzing the code, making the requests to SOCORA, and sending the email to the user with the result candidates. This class instantiates a *Cloner*(A.8, page 92) and a *DirectoryExplorer*(A.9, page 93) in the method *searchForComponent*. The former class is in charge of cloning the project to be analyzed and the latter is in charge of analyzing the code of the cloned project. Then the service *SocoraRequester*(A.16, page 101) is in charge of taking the information generated by *DirectoryExplorer* and make the request to SOCORA to search for components. This service, after getting the result from SOCORA, it sends an email with the candidates (represented by the class *Candidate*) to the recipient.
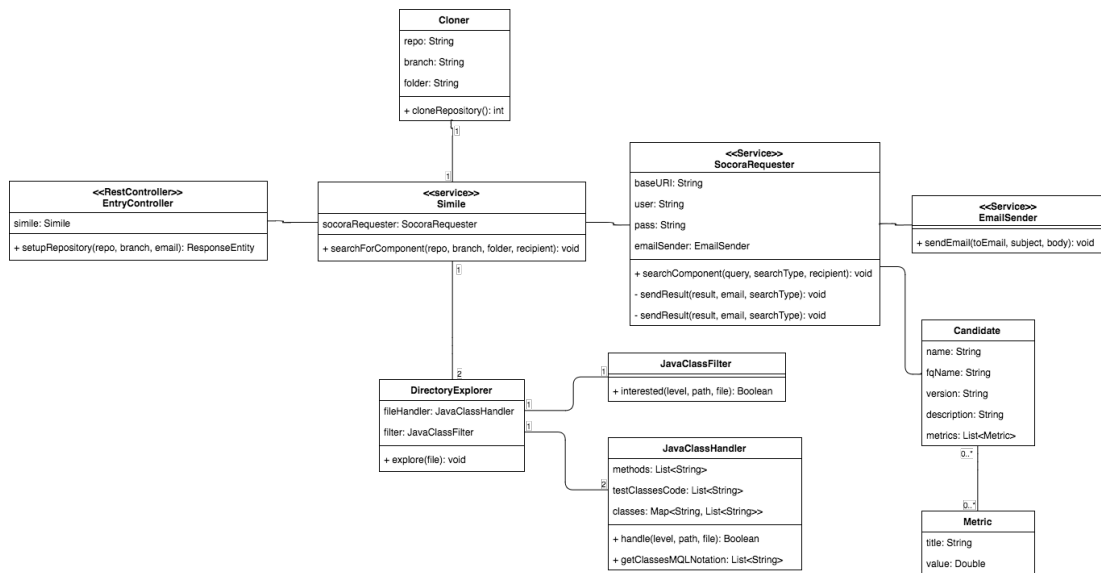
Figure 4.3.: Domain model

4.2. Implementation

In this section we will detail how the prototype was developed and the technologies used. First we will start explaining the Simile service which is the starting point of the whole process about cloning the project, analyse code, search components and send result. Then we will continue with the HTTP controller which receives the Git repository of the project to be analysed, the specific branch of the repository, and the email where the result will be sent. Then we will explain the Cloner object which is in charge of cloning the project locally. After that, we will explain how

the tool analyses the code and extracts the interface signatures and test classes
which will be used to make the requests to SOCORA. Next we will describe how
we receive the result from SOCORA. Finally we will explain how we handle the
result from the previous step and build the email that will be sent to user.

**NOTE: The code snippet shown in the following subsections has been
cleaned and it does not represent the source code. To see the source
code with all the details, please refer to the appendix A or to the CD
attached to this work.**

4.2.1. Simile

The class Simile.java [A.1] is a Spring service which contains only one method,
searchForComponents. This methods receives four parameters, repo, branch,
folder, and recipient. *repo* represents the Git repository of the project to be
analysed. *branch* is the branch of the Git repository (optional). *folder* represents
the name of the folder where the project will be cloned. Finally *recipient* is the
email where Simile will send the result. Listing 4.1 represents the method.

```
1   @Async
2   public void searchForComponents(
3       String repo,
4       String branch,
5       String folder,
6       String recipient) throws IOException, InterruptedException {
7
8     new Cloner(repo, branch, folder).cloneRepository();
9
10    JavaClassFilter javaClassFilter = new JavaClassFilter();
11
12    File maintDir = new File(String.format("%s/src/main", folder));
13    JavaClassHandler javaClassHandler = new JavaClassHandler();
14    new DirectoryExplorer(javaClassHandler, javaClassFilter).explore(
          maintDir);
15
16    File testDir = new File(String.format("%s/src/test", folder));
17    JavaClassHandler testClassHandler = new JavaClassHandler();
18    new DirectoryExplorer(testClassHandler, javaClassFilter).explore(
          testDir);
19
```

```
20    this.makeRequestWithClasses(
21      javaClassHandler.getClassesMQLNotation(),
22      recipient);
23    this.makeRequestWithTestClasses(
24      testClassHandler.getTestClassesCode(),
25      recipient);
26  }
27
28  private void makeRequestWithClasses(
29    List<String> classes, String recipientEmail) {
30    classes.forEach(c -> {
31      try {
32        socoraRequester
33          .searchComponent(
34            c,
35            SocoraRequester.INTERFACE_DRIVEN_SEARCH,
36            recipientEmail);
37      } catch (Exception ex) {
38        logger.error(ExceptionUtils.getStackTrace(ex));
39      }
40    });
41  }
42
43  private void makeRequestWithTestClasses(
44    List<String> testClasses, String recipientEmail) {
45    testClasses.forEach(tc -> {
46      try {
47        socoraRequester
48          .searchComponent(
49            tc,
50            SocoraRequester.TEST_DRIVEN_SEARCH,
51            recipientEmail);
52      } catch (Exception ex) {
53        logger.error(ExceptionUtils.getStackTrace(ex));
54      }
55    });
56  }
```

Listing 4.1: Method searchForComponents of Simile.java

First, it clones the project in the server by calling the method *cloneRepository* of Cloner [A.8]. Then it analyses the Java main code of the project cloned and

the test classes. When it analyses the code it extracts the interface signatures and the test classes, we explain how it does this process in section **??**. With the result of the previous step, we make the requests to SOCORA to search for similar components.

It is important to notice that this method is asynchronous by the Java annotation *@Async*. This allows us to run this method many times in parallel (configured in SimileApplication.java A.3).

4.2.2. Cloner

The class Cloner.java [A.8] is in charged of cloning the project in a local directory. The listing 4.2 represents the method *cloneRepository*.

```
1  public int cloneRepository() throws IOException {
2     int exitVal = 0;
3     try {
4        String command = this.setCommand(repo, branch, folder);
5        Process p = Runtime.getRuntime().exec(command);
6        StreamGobbler errorGobbler =
7                 new StreamGobbler(p.getErrorStream(), "INFO");
8        errorGobbler.start();
9        exitVal = p.waitFor();
10       return exitVal;
11    } catch (InterruptedException e) {
12       e.printStackTrace();
13    }
14    return exitVal;
15 }
```

Listing 4.2: Method cloneRepository of Cloner.java

This method runs the command *git clone* with the parameters *repo*, *branch*, and *folder*. The command is built by the private method *setCommand*. After running the command, the project is cloned locally under the folder with given *folder* name.

4.2.3. Interface signature and test class extraction

In this section we will explain how Simile explores the project cloned and extracts the interface signatures and test classes that will be sent to SOCORA to search

for similar components.

After the project is cloned locally by the Cloner (4.2.2), Simile goes through the code and extract interface signatures of each class, and test classes. The signatures extracted are transformed into Merobase Query Language (MQL) notation, and the test classes are extracted as is. The snipped code 4.3 shows how the interface signatures are extracted and transformed into MQL. For extracting test classes the principle is the same, the difference is that we differentiate a Test class from another Java file if it contains the annotation *@Test*.

```java
public void handle(int level, String path, File file) {
  try {
    JavaClassVisitor jcv = new JavaClassVisitor();
    jcv.visit(JavaParser.parse(file), null);
    jcv.getMethods()
      .forEach(method -> methods.add(
        this.getMethodMQLNotation(
        method.getNameAsString(),
        method.getParameters(),
        method.getType().toString()
        )
      ));
    classes.put(jcv.getClassObj().getNameAsString(), methods);
    methods = new ArrayList<>();
    if(jcv.getTestClasses().size() > 0)
      testClassesCode.addAll(jcv.getTestClasses());
  } catch (IOException e) {
    new RuntimeException(e);
  }
}

private String getMethodMQLNotation(
  String methodName,
  NodeList<Parameter> params,
  String returnType) {

  List<String> paramTypes = new ArrayList<>();
  params.forEach(param -> paramTypes.add(param.getType().toString())
    );
  return String.format(
    "%s(%s):%s",
    methodName,
```

```
32        StringUtils.join(paramTypes, ','),
33        returnType);
34  }
```

<div align="center">Listing 4.3: Method handle of JavaClassHandler.java</div>

In the class JavaClassVisitor.java we can see how Simile differentiate between a normal Java class and a test class. It implements the methods *visit* for a *ClassOrInterfaceDeclaration* and for a *MethodDeclaration* object of the abstract class VoidVisitorAdapter. The listing 4.4 show the implementation of these methods.

```
1   @Override
2   public void visit(ClassOrInterfaceDeclaration n, Object arg) {
3       super.visit(n, arg);
4       this.classObj = n;
5       if(n.toString().contains("@Test"))
6           testClasses.add(n.toString());
7   }
8
9   @Override
10  public void visit(MethodDeclaration n, Object arg) {
11      super.visit(n, arg);
12      methods.add(n);
13  }
```

Listing 4.4: Implementation of methods *visit* of VoidVisitorAdapter in JavaClassVisitor.java

4.2.4. SOCORA Integration

Our prototype uses SOCORA to search for components. After extracting the interface signatures and test classes from the project, Simile make requests to SOCORA using this information. SOCORA then searches for similar components using Merobase and ranks the candidates using non-functional requirements. Once it is done, SOCORA sends the result back to Simile.

Simile communicates with SOCORA by making HTTP requests using the interface signatures and the test classes extracted. SOCORA support several query parameters such as filters and ranking criteria. Our tools supports some of them by default depending on the type of search. For textual search our prototype support the following query parameters:

- Filters

  - Entry class hashcode clone: Filters out additional entry class hash clones

  - No abstract class: Filters out abstract classes

  - No interface: Filters out interfaces

  - Latest version: Filters Maven-based components based on later versions

  - Functional sufficiency

- Ranking Strategy

  - Single objective

- Ranking criteria

  - Lucene score - objective: maximize

- Others

  - Max number of results = 10

For test-driven search Simile uses the following query parameters by default:

- Filters

  - Entry class hashcode clone: Filters out additional entry class hash clones.

  - No abstract class: Filters out abstract classes.

  - No interface: Filters out interfaces.

  - Latest version: Filters Maven-based components based on later versions.

  - Functional sufficiency: Filters out components that do not meet functional requirements.

- Ranking Strategy

  - Hybrid non-dominated sorting

- Non-functional metrics

- Leanness default: Leanness based on instructions count (Needed Functionality/Necessary Functionality, [0, 1] - objective: maximize

- Throughput operations per second - objective: maximize.

- Cohesion LCOM3: LCOM3 cohesion (CK), scope entry class - objective: minimize

- Efferent Coupling: Efferent Coupling of Entry Class (How many other classes are used by this class?) - objective: minimize.

- Others

   - Max number of results = 400

Listing 4.5 represents the implementation of textual search request and Listing 4.6 represents the test-driven search request.

```
1  private void textualSearchComponent(
2    String method,
3    String recipient) throws IOException, TemplateException {
4
5    CandidateSearchClient candidateSearchClient =
6      new CandidateSearchClient(baseURI, auth(user, pass));
7    CandidateSearchRequest request =
8      new CandidateSearchRequest();
9
10   int maxResults = 10;
11
12   request.setInput(method);
13   QueryParams queryParams = new QueryParams();
14   queryParams.setRows(maxResults);
15
16   queryParams.setArtifactInformation(true);
17   queryParams.setContent(true);
18
19   // FILTERS
20   queryParams.setFilters(Arrays.asList(
21     Filters.HASH_CODE_CLONE_FILTER,
22     Filters.NO_ABSTRACT_CLASS_FILTER,
23     Filters.NO_INTERFACE_FILTER,
24     Filters.LATEST_VERSION_FILTER,
25     Filters.FUNCTIONAL_SUFFICIENCY_FILTER
26   ));
```

```
27
28    request.setQueryParams(queryParams);
29    request.setTestDrivenSearch(Boolean.FALSE);
30
31    queryParams.setRankingStrategy(Rankings.SINGLE_OBJECTIVE);
32
33    List<RankingCriterion> rankingCriteria = Collections.singletonList
         (
34      RankingCriterionBuilder
35        .rankingCriterion()
36        .withName("luceneScore")
37        .withObjective(RankingCriterion.MAX)
38        .build()
39    );
40
41    queryParams.setRankingCriteria(rankingCriteria);
42
43    CandidateSearchResponse response =
44      candidateSearchClient.search(request);
45    CandidateListResult result =
46      response.getCandidates();
47
48    this.sendResult(
49      this.processTemplateTextualSearchResult(method, result,
         queryParams),
50      recipient,
51      "Textual search"
52    );
53 }
```

Listing 4.5: Method textualSearchComponent of SocoraRequester.java

```
1  private void testDrivenSearchComponent(String testClassSourceQuery,
      String recipient) throws IOException, InterruptedException,
      TemplateException {
2    CandidateSearchClient candidateSearchClient = new
         CandidateSearchClient(baseURI, auth(user, pass));
3    CandidateSearchRequest request = new CandidateSearchRequest();
4
5    int maxResults = 400;
6
7    request.setInput(testClassSourceQuery);
8    QueryParams queryParams = new QueryParams();
```

```
 9    queryParams.setRows(maxResults);
10
11    queryParams.setArtifactInformation(true);
12    queryParams.setContent(true);
13
14    // FILTERS
15    queryParams.setFilters(Arrays.asList(
16      Filters.HASH_CODE_CLONE_FILTER,
17      Filters.NO_ABSTRACT_CLASS_FILTER,
18      Filters.NO_INTERFACE_FILTER,
19      Filters.LATEST_VERSION_FILTER,
20      Filters.FUNCTIONAL_SUFFICIENCY_FILTER
21    ));
22
23    request.setQueryParams(queryParams);
24
25    request.setTestDrivenSearch(Boolean.TRUE);
26
27    queryParams.setRankingStrategy(Rankings.
         HYBRID_NON_DOMINATED_SORTING);
28
29    RankingCriterion fsCriterion = new RankingCriterion();
30    fsCriterion.setName("sf_instruction_leanness");
31    fsCriterion.setObjective(RankingCriterion.MAX);
32    RankingCriterion performanceCriterion = new RankingCriterion();
33    performanceCriterion.setName("jmh_thrpt_score_mean");
34    performanceCriterion.setObjective(RankingCriterion.MAX);
35    RankingCriterion lcomCriterion = new RankingCriterion();
36    lcomCriterion.setName("entryClass_ckjm_ext_lcom3");
37    lcomCriterion.setObjective(RankingCriterion.MIN);
38    RankingCriterion couplingCriterion = new RankingCriterion();
39    couplingCriterion.setName("entryClass_ckjm_ext_ce");
40    couplingCriterion.setObjective(RankingCriterion.MIN);
41
42    queryParams.setRankingCriteria(Arrays.asList(
43      fsCriterion,
44      performanceCriterion,
45      lcomCriterion,
46      couplingCriterion
47    ));
48
49    Stopwatch timer = Stopwatch.createStarted();
```

```
50    CandidateSearchResponse response = candidateSearchClient.search(
          request);
51
52    String jobId = response.getJobId();
53
54    JobStatus jobStatus = null;
55    while (jobStatus == null ||
56      jobStatus.getStatusType() != JobStatusType.FINISHED) {
57
58      jobStatus = candidateSearchClient.getJobStatus(jobId);
59      Thread.sleep(30 * 1000L);
60    }
61
62    if(jobStatus.getStatusType() == JobStatusType.FINISHED) {
63      timer.stop();
64      getTestDrivenSearchResult(jobId, recipient, testClassSourceQuery
          );
65    }
66 }
```

Listing 4.6: Method testDrivenSearchComponent of SocoraRequester.java

The class in charge of making the request to SOCORA is *SocoraRequester.java*[A.16]. The method *searchComponent* receives two parameters, *queries* which represents the interface signatures or the test classes, and *searchType* which represents the type of search (i.e. test driven search, interface driven search or textual search (default)). The snippet of code 4.7 represent the method *searchComponent*.

```
1  public static String TEST_DRIVEN_SEARCH = "TEST_DRIVEN_SEARCH";
2  public static String INTERFACE_DRIVEN_SEARCH = "
       INTERFACE_DRIVEN_SEARCH";
3  public static String KEYWORD_SEARCH = "KEYWORD_SEARCH";
4
5  @Autowired
6  private EmailSender emailSender;
7
8  public void searchComponent(
9    String query,
10   String searchType,
11   String recipient) throws Exception {
12
13   Validate.notBlank(query, "Query parameter is required and cannot
```

```
          be blank");
14    if (StringUtils.compare(searchType, INTERFACE_DRIVEN_SEARCH) == 0
         ||
15        StringUtils.compare(searchType, KEYWORD_SEARCH) == 0 ||
16        StringUtils.isBlank(searchType)) {
17      this.textualSearchComponent(query, recipient);
18    } else if (StringUtils.compare(searchType, TEST_DRIVEN_SEARCH) ==
         0) {
19      this.testDrivenSearchComponent(query, recipient);
20    } else {
21      this.textualSearchComponent(query, recipient);
22    }
23  }
```

Listing 4.7: Method searchComponent of SocoraRequester.java

When SOCORA responses with the list of components found they are sent by
email to developers. This action is done in the methods *textualSearchComponent*
or *testDrivenSearchComponent* using the service EmailSender [A.10].

4.2.5. Email result

After making the requests to SOCORA 4.2.4, it will response with the candi-
date components found. Simile receives the list of candidates, it extracts the
candidates from the response, builds the email, and sends it to the recipient.
For textual search results, the method *processTemplateTextualSearchResult* is in
charge of interpreting the result and process the email's template before it is sent.
For test-driven results, the method *processTemplateTestDrivenSearchResult* is in
charge of interpreting the result and process the email's template before it is sent.
The listing 4.8 show this method.

```
1  private String processTemplateTestDrivenSearchResult(
2    CandidateListResult result,
3    QueryParams queryParams,
4    String jobId) throws IOException, TemplateException {
5
6    Template emailTmp = freeMarker.getTemplate("testdrivenResultEmail.
         ftl");
7    StringWriter stringWriter = new StringWriter();
8    List<Candidate> candidates = this.extractSearchCandidates(result,
         queryParams);
```

```java
9     Map<String, Object> root = new HashMap<>();
10    root.put("jobId", jobId);
11    root.put("numMetrics", candidates.get(0).getMetrics().size());
12    root.put("metrics", candidates.get(0).getMetrics());
13    root.put("candidates", candidates);
14    emailTmp.process(root, stringWriter);
15
16    return stringWriter.toString();
17  }
18
19  private String processTemplateTextualSearchResult(
20    String methodQueried,
21    CandidateListResult result,
22    QueryParams queryParams) throws IOException, TemplateException {
23
24    Template emailTmp = freeMarker.getTemplate("interfaceResultEmail.
         ftl");
25    StringWriter stringWriter = new StringWriter();
26    List<Candidate> candidates =
27      this.extractSearchCandidates(result, queryParams);
28    Map<String, Object> root = new HashMap<>();
29    root.put("query", methodQueried);
30    root.put("numMetrics", candidates.get(0).getMetrics().size());
31    root.put("metrics", candidates.get(0).getMetrics());
32    root.put("candidates", candidates);
33    emailTmp.process(root, stringWriter);
34
35    return stringWriter.toString();
36  }
37
38  private List<Candidate> extractSearchCandidates(
39    CandidateListResult result,
40    QueryParams queryParams) {
41
42    List<Candidate> candidates = new ArrayList<>();
43    result.getCandidates().forEach(doc -> {
44      Candidate candidate = new Candidate();
45      candidate.setName(
46        Optional.ofNullable(doc.getArtifactInfo().getName()).orElse(""
             )
47      );
48      candidate.setFqName(
```

```
49        Optional.ofNullable(doc.getFQName()).orElse("")
50      );
51      candidate.setMetrics(
52        extractMetrics(doc.getMetrics(), queryParams.
            getRankingCriteria())
53      );
54      candidate.setDescription(
55        Optional.ofNullable(
56          doc.getArtifactInfo()
57            .getDescription()
58            .orElse("")
59      );
60      candidate.setVersion(
61        Optional.ofNullable(doc.getVersion()).orElse(""));
62      candidates.add(candidate);
63    });
64    return candidates;
65  }
```

Listing 4.8: Methods   *processTemplateTextualSearchResult*   and
            processTemplateTestDrivenSearchResult of SocoraRequester.java

Figure 4.4.: Textual search result email

After the email's body is generated by using the template and the java objects,

it is sent to the recipient. The figures 4.4 and 4.5 show the email with the result of a textual search and a test-driven search respectively.

Figure 4.5.: Test-driven search result email

Depending on the type of search, the email will have some differences. The text driven search result contains a the query sent to SOCORA and a table with the candidates found. For test-driven search result, the email contains the job id of the search, the link to SOCORA to see more details of the result and the list of candidates found.

4.2.6. HTTP Controller

The Java class EntryController.java [A.4] is in charge of receiving the request from a customer with the Git repository and the branch of the project to be analysed, and the email where the result of the similar component will be sent. In the listing 4.9 we can see the method *setupRepository* which represents the entry point which is a POST request with the three parameters described before. The parameter *repo* is required and represent the Git repository of the project that will be analysed and to which similar components will be searched for. The parameter *branch* represents the branch of the git repository, this parameter is

optional. Finally, the parameter *email* represents the email where the result of
the similar components will be sent to.

```java
@RequestMapping(name = "/repository", method = RequestMethod.POST)
public ResponseEntity<?> setupRepository(
  @RequestParam(name = "repo") String repo,
  @RequestParam(
    name = "branch",
    defaultValue = "master",
    required = false) String branch,
  @RequestParam(name = "email") String email) throws Exception {

  if(validateRequest(repo, email).isPresent()) {
    return ResponseEntity
          .badRequest()
          .body(validateRequest(repo, email)
          .get());
  } else {
    simile.searchForComponents(repo, branch, Cuid.createCuid(),
        email);
    return ResponseEntity.ok(new Message("Repository set
        successfully", 200));
  }
}
```

Listing 4.9: Method setupRepository of EntryController.java

When a request is received by the controller, it firstly validates if the parameters
are correct. If they are not valid, it will respond with HTTP status code 400 and a
message describing the errors. Otherwise, it triggers asynchronously the method
searchForComponents with the repository, the branch, and a random CUID as a
folder name of the service Simile. This service is in charge of starting the process
of cloning the project, analyse the code, make the request to SOCORA, and to
send the email with the result to the customer.

4.3. CI integration

To integrate our prototype to a Continuous Integration process we decided to
create a plugin. We decided to create our main prototype as independent as
possible thus we do not depend on any CI tool. Therefore in this work to prove

our approach we built a plugin for the CI server Jenkins. We decided to use Jenkins because is one of the most popular CI server for Java projects [1] and it is open-source.

The Simile Jenkins plugin is simple. First when a user creates a job in Jenkins, he/she can add a post build step. There he/she select the Simile plugin. When it is done, a new section is added in the task configuration (figure 4.6). There he/she should add the Git repository of the project, the branch (optional), and the email. Once done, whenever the job is triggered, the plugin will be triggered too. The plugin sends via HTTP POST request the repository, branch, and email to Simile.
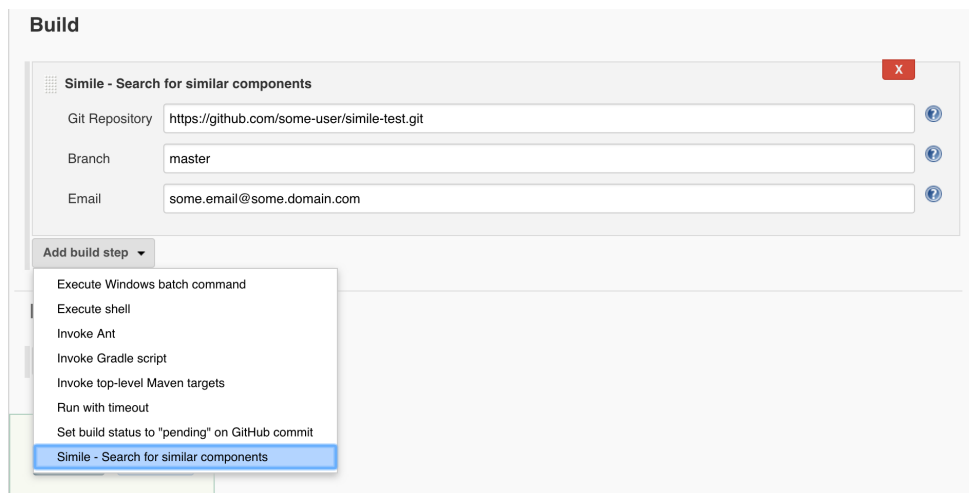
Figure 4.6.: Simile Jenkins plugin configuration in Jenkins task

To configure the endpoint where Simile is deployed, we just need to go to global settings and set the URL. Figure 4.7 depicts this option.

In the future if Simile needs to be integrated to another CI server, we just need to develop a simple plugin which sends the information needed to look for similar components.

---

[1] `https://www.cloudbees.com/sites/default/files/2016-jenkins-community-survey-responses.pdf` (accessed: 04.02.2017)

Figure 4.7.: Simile endpoint configuration in Jenkins global configuration

As a conclusion in Figure 4.8 we can appreciate how the objects interact with each other. The process begins with the method *searchRepository* which contains the repository of the project, the branch and the email where the result will be sent. It triggers the method *searchForComponent* of the service Simile asynchronously. Right after the project is cloned locally by Cloner object and it is explored by DirectoryExplorer object. After the classes, its methods, and test classes are extracted from the project, we get into two loops. In the first loop, for each Java class extracted Simile makes a request to SOCORA. The query is the class extracted defined in MQL notation. Once the result set is returned by SOCORA, Simile prepares the email with the candidates and sends them to the recipient.
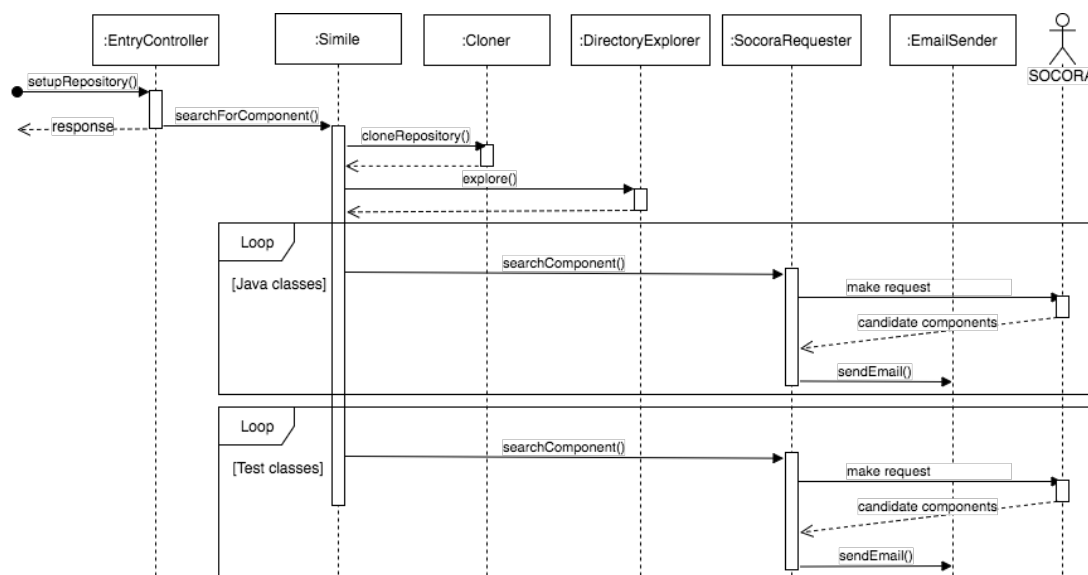
Figure 4.8.: Sequence diagram

4.4. Search Results

In this section we will evaluate the results obtain using Simile in our test-project. To test our proof-of-concept we used a project that implements the usage scenario we explain in Chapter 3. First, we deployed Jenkins in a Tomcat 9 instance and we installed the Simile Jenkins Plugin. After that we created a task in Jenkins and we configured the plugin following the steps described in Section 4.3. Then we ran the task to get the candidates for component reuse. Finally we received the result and measured the number of candidates returned by SOCORA, non-functional requirements, and time taken to return the results.

4.4.1. Textual search

For textual search, Simile takes the classes and the method it contains, and then transform them into MQL notation. Our test project has only one class *Base64.java* (Listing 3.1, page 37) which contains two methods: *encode* and *decode.* The MQL notation of this class is the following:

Base64(encode(byte[]):char[];decode(char[]):byte[];)

By default the application used the following query parameters:

- Filters
    - Entry class hashcode clone
    - No abstract class
    - No interface
    - Latest version
    - Functional sufficiency
- Ranking Strategy
    - Single objective
- Ranking criteria
    - Lucene score - objective: maximize
- Others

Component	FQ Name	Ranking Criteria Lucene Score	Version
Apache Tuscany SCA Binding WS Runtime Axis2 Aggregation	org.apache.axiom.om.util.Base64	9.73	2.0.1
Apache ServiceMix Bundles	org.apache.axiom.om.util.Base64	9.73	1.2.13-1
REST API SDK	com.paypal.base.codec.binary.Base64	9.61	1.3.0
Ajax4JSF	org.ajax4jsf.framework.util.base64.Base64	8.97	1.0.6
jose4j	org.jose4j.base64url.internal.apache.commons.codec.binary.Base64	8.43	0.4.4
Lib-Cross-Java JSE5	com.github.libxjava.io.Base64	7.98	0.3
Lib-Cross-Java CDC	com.github.libxjava.io.Base64	7.98	0.3
Lib-Cross-Java CLDC	com.github.libxjava.io.Base64	7.98	0.3
rfc-4648	com.googlecode.jinahya.rfc4648.Base64	7.96	1.0.2
WildFly: Process Controller	org.jboss.as.process.stdin.Base64	7.64	8.2.1.Final

Table 4.1.: Textual search result

      − Max number of results = 10

Using the MQL notation of the class and the parameters, Simile makes the request to SOCORA. The code can be found in class *SocoraRequester.java* in method *textualSearchComponent* (Listing A.16, page 101).

Table 4.1 details the result returned by SOCORA. The candidates are sorted by Lucene score, the higher the number more similar is the result to the query. Therefore the first candidate, org.apache.axiom.om.util.Base64[2], is more similar to the query than the last candidate, org.jboss.as.process.stdin.Base64[3]. If we check the code of these candidates, we can see that the first one actually implements the two methods of the query using the same input but with different output. Whereas the last one implements the methods but with different inputs and outputs.

The time taken from making the request to SOCORA to receive the result was 6.07 seconds[4].

---

[2]http://grepcode.com/file/repo1.maven.org/maven2/org.apache.ws.commons.axiom/axiom-api/1.2.15/org/apache/axiom/util/base64/Base64Utils.java (accessed: 21.02.2017)

[3]https://github.com/wildfly/wildfly-core/blob/master/process-controller/src/main/java/org/jboss/as/process/stdin/Base64.java (accessed: 21.02.2017)

[4]This time does not consider the time taken to receive the email with the result.

4.4.2. Test-driven search

For test-driven search, Simile takes the test classes that the project might have. Our test project contains only one test class, *Base64Test.java* (Listing 3.2, page 38), which contains one test and two auxiliary methods. This test is taken by Simile as is and it is used as query.

By default the the query parameters for test-driven search are the following:

- Filters

    - Entry class hashcode clone[5]

    - No abstract class[6]

    - No interface[7]

    - Latest version[8]

    - Functional sufficiency

- Ranking Strategy

    - Hybrid non-dominated sorting

- Non-functional metrics

    - Leanness default[9] - objective: maximize

    - Throughput operations per second - objective: maximize

    - Cohesion LCOM3[10] - objective: minimize

    - Efferent Coupling[11] - objective: minimize

- Others

    - Max number of results = 400

---

[5]Filters out additional entry class hash clones

[6]Filters out abstract classes

[7]Filters out interfaces

[8]Filters Maven-based components based on later versions

[9]Leanness based on instructions count (Needed Functionality/Necessary Functionality, [0, 1]. Default objective is to maximize

[10]LCOM3 cohesion (CK), scope entry class

[11]Efferent Coupling of Entry Class (How many other classes are used by this class?)

Component	FQ Name	Metrics			Version
		Leanness Default	Efferent Coupling	Cohesion LCOM3	
OW2 Util :: Base64	org.ow2.util.base64.Base64	1	6	1.33	1.0.37
scalaj-http	scalaj.http.Base64	0.69	15	1.08	2.0.0
Interfaces	org.imixs.workflow.webservices.Base64	0.55	22	1.13	2.1.1
jtstand-common	org.jfree.xml.util.Base64	0.56	22	1.13	1.5.9
Macaroons: Cookies	com.github.nitram509.jmacaroons.util.Base64	0.95	6	1.13	0.3.1
Core API	org.imixs.workflow.util.Base64	0.55	22	1.13	3.4.4
jcommon	org.jfree.xml.util.Base64	0.56	22	1.13	1.0.16
TrAP Utils API	com.ericsson.research.trap.utils.Base64	0.46	16	1.08	1.4.1
jbundle-thin-base-util-base64 - base64 code	org.jbundle.thin.base.util.base64.Base64	0.1	9	1.14	1.0.9
JASMINe :: Self management :: Rules :: Cluster :: JK :: Common	org.ow2.jasmine.rules.cluster.jk.Base64	1	6	1.33	1.3.7
JPush API Java Client	cn.jpush.api.utils.Base64	1	5	1.33	3.2.7
Ganymed SSH2 for Java	com.trilead.ssh2.crypto.Base64	1	5	1.33	build212-hudson-6
ganymed-ssh-2	ch.ethz.ssh2.crypto.Base64	1	5	1.33	262
Ganymed SSH2 for Java	ch.ethz.ssh2.crypto.Base64	1	5	1.33	build210-hudson-1
JOnAS :: Libraries :: Commons	org.objectweb.jonas.common.encoding.Base64	1	6	1.33	5.0-M1
jyal	com.github.to2mbn.jyal.util.Base64	1	4	1.25	1.3
scrypt	com.lambdaworks.codec.Base64	0.98	5	1.17	1.4.0
codec	com.lambdaworks.codec.Base64	0.98	5	1.17	1.0.0
OW2 Utilities :: Base64	org.ow2.util.base64.Base64	1	6	1.33	2.0.0

Table 4.2.: Test-driven search result

Using the test class and the query parameters, Simile makes the request to SO-CORA. The code can be found in class *SocoraRequester.java* in method *testDrivenSearchComponent* (Listing A.16, page 101).

Table 4.2 details the result given by SOCORA for our test class with the query parameters. All of these candidates meet the functional requirements, so all of them might be reused. The difference is in the metrics. Neither the first component of the list is the best nor the last one is the worst, that is defined by the user's requirements. For this reason we include the link to the SOCORA webpage where the user can visualize this result and sort them based on its requirements. For instance, for one user might be more important the leanness value rather than the other, thus he/she would be more interested on the components with highest leanness value.

The time taken from making the request to SOCORA to receive the result was 45 minutes and 46.306 seconds (00:45:46.306)[12].

---

[12]This time does not consider the time taken to receive the email with the result.

Although the time taken to return a result using test-driven search is quite high, this is not necessarily a complication. This is due to the fact that this search is made in the background and the developer can keep working until he/she receives the result by email.

4.5. Technology

In this final section of this chapter, we will describe the different technologies used to develop our proof-of-concept.

JavaParser   is a library that parses Java code and creates an Abstract syntax tree (AST[13]), which records the source code structure, javadoc and comments. Moreover, this library allows to change the AST nodes or create new ones to modify the source code[14].

Spring   is a framework that provides a comprehensive programming and configuration model for modern Java-based enterprise applications. A key element of Spring is its infrastructural support at the application level: Spring focuses on the *plumbing* of enterprise applications so that teams can focus on application-level business logic, without unnecessary ties to specific deployment environments[15].

Spring Boot   is an opinionated instance of a Spring application. Spring Boot is a rapid application development platform. It uses various components of Spring, but has additional features like the ability to package your application as a runnable jar, which includes an embedded tomcat (or jetty) server[16].

Guava   is a set of core libraries developed by Google that includes new collection types (such as multimap and multiset), immutable collections, a graph library,

---

[13]Abstract syntax tree (AST): abstract syntax tree (AST), or just syntax tree, is a tree representation of the abstract syntactic structure of source code written in a programming language.

[14]JavaParser - `http://javaparser.org/` (accessed: 21.02.2017)

[15]Spring - `http://projects.spring.io/spring-framework/` (accessed: 20.02.2017)

[16]SpringBoot - `http://projects.spring.io/spring-boot/` (accessed: 20.02.2017)

functional types, an in-memory cache, and APIs/utilities for concurrency, I/O, hashing, primitives, reflection, string processing, among other features[17].

CUID-Java   is a small library written in Java which generates collision-resistant ids[18]. We use this tool for generating a random ID as a folder name when the application clones a project. Thus we avoid the risk of creating duplicated ids.

FreeMarker   is a template engine which generates text output (HTML web pages, e-mails, configuration files, source code, etc.) based on templates and changing data[19].

Tomcat 9   is an open-source container that implements the Servlet 4.0 and JavaServer Pages 2.3 specifications[20]. We use Tomcat to deploy Jenkins and Simile for local testing.

---

[17]Guava -`https://github.com/google/guava` (accessed: 20.02.2017)
[18]CUID: Collision-resistant ids - `https://github.com/graphcool/cuid-java` (accessed: 04.02.2017).
[19]`http://freemarker.org/` (accessed: 16.02.2017)
[20]Apache Tomcat 9 - `http://tomcat.apache.org/tomcat-9.0-doc` (accessed: 21.02.2017)

5. Discussion

The first question that arose during this work is: *Is this approach beneficial?*. As a first attempt to improve software reuse by using continuous integration, it is difficult to answer the question. In fact, it is more likely that at this version it might not be very helpful due to its immaturity. However, it is very promising in the future. In this chapter we will discuss about the possible pros and cons of this work.

This approach reduces the effort of searching suitable components by analyzing the code in background. Simile was conceived as unobtrusive tool as possible, that is why we found a great opportunity in the continuous integration process and the agile technique TDD. When a project is developed using TDD, the tests are written before the code that need to be implemented. Also according to the CI workflow, developers should commit changes as often as possible which trigger the CI server to validate the changes by testing the code. At this moment we visualized a great chance to look for similar components using textual and test-driven search approaches. Whenever the CI server is triggered we have the chance to analyze the code, extract interface signatures, and test classes which are sent to the code-search engine to retrieve similar components to the components that are being implemented. Thus developers might be able to reduce risk and development cost by reusing components which implement the functionality they are working on. Moreover it reduces the effort needed for searching components by running in background without disturbing the work of the developers.

Our prototype is able to provide high precision result that fully meet functional requirements. As test-driven search technique is able to provide the best results from the semantic point of view [48, 26, 24], we can assure that the precision of the components returned is very high (depending on the quality of the tests). We achieve this by relying on Merobase as code-search engine.

Although Simile can provide high precision on the components retrieved, if these are not ranked accordingly the effort invested on reusing components would in-

crease. We solve this problem by relying on SOCORA which is able to rank components using combination of non-functional properties such as performance or maintenance [30].

The future of Simile appear to be attractive due to the massive adoption of CI/CD. Continuous Integration/Delivery adoption rates is showing an unexpected increment. According to the research made by Perforce Software [52], 65% of companies have adopted CI/CD in some or all projects. Moreover, 40% of open-source projects in Github have also adopted Continuous Integration [20]. This is a great opportunity for this tool to be used vastly by companies and open-source projects, helping them to reduce development costs and time.

This approach also has some possible cons. If a team does not use TDD or any test, Simile would only be able to extract interface signatures from the code. Therefore the result retrieved would not be as precise as it would be using tests. This is due to the fact that the search would be based on keyword and text-matching techniques which do not necessarily meet functional requirements.

Another cons is the high time it might takes to deliver a result. Test-driven search component is heavy-weight task because it requires to run tests to for each possible candidate. Although it has been improved by parallelization of test execution and optimizing the underlying code-search engine, it still takes a reasonable time for returning a result. This time can be even bigger depending on the complexity of the tests used for searching components.

One more cons is the bad quality of the documentation of the candidate components. Although the components retrieved meet the functional requirements of the user, if they are poorly documented or they are hard to understand the effort invested on reusing them will be high. Therefore it would be helpful to consider the quality of the code and documentation of the components retrieved.

6. Future Work

In this work we presented a novel approach which improves the chances of reusing software by seizing continuous integration process. The approach seems promising as developers do not need to spend much effort on searching reusable component as the prototype, Simile, does this automatically in the background. However, there still exist many aspects with potential for improvement.

Currently our proof-of-concept supports only projects developed in Java, so it would be interesting to support other programming languages such as Javascript, Scala, or Go.

Support other CI servers and VCSs is a task to be done in the future. In the ongoing version of Simile, it supports only Jenkins as CI server therefore the support of other CI software however to support other alike tools is not a difficult task. This is due to the fact that it is only necessary to implement plugins with the functionality of send information to Simile. Nevertheless to support other VCSs is more complex as it will require changes in the core of Simile.

Although the time taken by code-search engines and rankings to deliver a result have been reduced, it is still considerable to make test-driven reuse applicable for the daily work of a developer.

Further testing must be done in order to evaluate the usefulness of the approach. For our proof-of-concept we used a dummy project which implements Base64 as described in Chapter 3. Therefore it would be interesting to apply Simile in a real project to analyze information. For instance data like number of components retrieved, number of components reused, or time taken to retrieve a result would be interesting to measure.

Enhance the features provided by this prototype is another task to be considered. Currently the request to SOCORA for searching component using test-driven approach supports three non-functional requirements: leanness, operations/second, cohesion, and efferent coupling. These are by default and they cannot be change

by the user. Therefore, for a future version more non-functional requirements should be supported such as superfluous functionality, needed functionality, balance, functional sufficiency, among others. Send the result to more emails at the same time is another feature to be implemented from now on.

Despite the fact that SOCORA, which our proof-of-concept relies on for searching components, is in a work-in-process status, the team behind is working on enhancing features provided. Integration into desktop IDEs, inclusion of of further metrics and quality models specially tailored to pragmatic reuse, the accommodation of more relaxed filtering criteria in the result set and providing guidance on the nature of the test that should be supplied as search queries, are some of the features planed to be implemented in SOCORA[30].

7. Conclusion

In this work we have presented a novel approach which is an attempt to improve the chances of reusing software components by seizing continuous integration process. As a proof-of-concept we implemented Simile, which is a service that analyzes code, extracts interface signatures and test classes, and send them to SOCORA for searching similar components. Once SOCORA responses with the components found, this tool sends an email with the details of the candidates to the recipient.

Several benefits come with this approach. Development cost and time development can be reduce by reusing software components, however some human-related challenges might jeopardize these benefits. For instance developers need to be aware of the existence of components that be suitable to its needs. For this reason, the approach presented in this work is a first step to improve the success of software reuse practice in a project. This is due to the fact that Simile is able to work in parallel of the software development process, thus it is able to recommend components with almost no effort from the developers to find and evaluate candidate reusable components. Moreover, by using test-driven search and SOCORA ranking, this tool can recommend components that fully meet the functional requirements ranked based on non-functional requirements.

Despite the fact that this approach is a contribution, it is still immature and needs further improvements. First, our proof-of-concept supports only Jenkins, so other popular CI servers should be supported in the future. Second, supporting the selection of other non-functional requirements by the user is need to be implemented. Third, test-driven search approach necessitates further refinements to improve the time taken to return a result. Finally, SOCORA is a on-going project that will be improved in future version by supporting more features which will improve the outcomes presented in this document.

Bibliography

[1] Scott Ambler. Introduction to Test Driven Development (TDD). http://agiledata.org/essays/tdd.html, 2013. URL `http://agiledata.org/essays/tdd.html`. Accessed: 21.02.2017.

[2] Veronika Bauer and Antonio Vetro'. Comparing reuse practices in two large software-producing companies. 117:545–582, 2016. ISSN 01641212. doi: 10.1016/j.jss.2016.03.067.

[3] Kent Beck. *Extreme Programming Explained: Embrace Change*. Number c. 1999. ISBN 0201616416. doi: 10.1136/adc.2005.076794.

[4] Kent Beck. Test-Driven Development By Example. 2(c):176, 2003. ISSN 1660-1769. doi: 10.5381/jot.2003.2.2.r1.

[5] Grady Booch, Robert a. Maksimchuk, Michael W. Engle, Bobbi J. Young, Jim Conallen, and Kelli a. Houston. *Object-Oriented Analysis and Design with Applications*, volume 1. 2007. ISBN 020189551X. doi: 10.1145/1402521.1413138.

[6] Scott Chacon. Pro Git. pages 1–210, 2009. ISSN 978-1-4302-1833-3. doi: 10.1007/978-1-4302-1834-0.

[7] Lianping Chen. Towards Architecting for Continuous Delivery. In *2015 12th Working IEEE/IFIP Conference on Software Architecture*, pages 131–134. IEEE, may 2015. ISBN 978-1-4799-1922-2. doi: 10.1109/WICSA.2015.23. URL `http://ieeexplore.ieee.org/document/7158514/`.

[8] George Dinwiddie. If you don't automate acceptance tests? http://blog.gdinwiddie.com/2009/06/17/if-you-dont-automate-acceptance-tests/, 2009. URL `http://blog.gdinwiddie.com/2009/06/17/if-you-dont-automate-acceptance-tests/`. Accessed: 21.02.2017.

[**9**] Paul Duvall, Steve Matyas, and Andrew Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. 2007. ISBN 9780321336385. URL `http://portal.acm.org/citation.cfm?id=1406212`.

[**10**] The Apache Software Foundation. Apache Lucene - Apache Lucene Core. https://lucene.apache.org/core/, 2017. Accessed: 12.01.2017.

[**11**] Martin Fowler. Continuous Integration. http://martinfowler.com/articles/continuousIntegration.html, 2006. Accessed: 11.09.2016.

[**12**] Martin Fowler. Polyglot Persistence. https://martinfowler.com/bliki/PolyglotPersistence.html, 2011. Accessed: 03.02.2017.

[**13**] Martin Fowler. Continuous Delivery. https://martinfowler.com/bliki/ContinuousDelivery.html, 2013. Accessed: 09.01.2017.

[**14**] Martin Fowler and James Lewis. Microservices. https://martinfowler.com/articles/microservices.html, 2014.

[**15**] W.B. Frakes and Kyo Kyo Kang. Software reuse research: status and future. 31(7):529–536, jul 2005. ISSN 0098-5589. doi: 10.1109/TSE.2005. 85. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1492369`.

[**16**] William Frakes and Carol Terry. Software reuse: metrics and models. 28 (2):415–435, jun 1996. ISSN 03600300. doi: 10.1145/234528.234531. URL `http://portal.acm.org/citation.cfm?doid=234528.234531`.

[**17**] Ned Freed and Nathaniel Borenstein. Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies. RFC 2054, The Internet Engineering Task Force (IETF), November 1996. URL `https://www.ietf.org/rfc/rfc2045.txt`.

[**18**] Vinicius C. Garcia, Eduardo S. de Almeida, Liana B. Lisboa, Alexandre C. Martins, Silvio R. L. Meira, Daniel Lucredio, and Renata P. de M. Fortes. Toward a Code Search Engine Based on the State-of-Art and Practice. In *2006 13th Asia Pacific Software Engineering Conference (APSEC'06)*, pages

61–70. IEEE, 2006. ISBN 0-7695-2685-3. doi: 10.1109/APSEC.2006.57. URL `http://ieeexplore.ieee.org/document/4137403/`.

[**19**] Sarah Goff-Dupont. Why every development team needs continuous delivery. http://blogs.atlassian.com/2015/10/why-continuous-delivery-for-every-development-team, 2015. Accessed: 21.02.2017.

[**20**] Michael Hilton, Timothy Tunnell, Darko Marinov, Danny Dig, 1978 Huang, Kai, Oregon State University. School of Electrical Engineering Science, and Computer. Usage, Costs, and Benefits of Continuous Integration in Open-Source Projects. 2016.

[**21**] Bradley Horowitz. Official Google Blog: A fall sweep. https://googleblog.blogspot.cl/2011/10/fall-sweep.html, 2011. Accessed: 12.01.2017.

[**22**] Jezz Humble and David Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation.* 2010. ISBN 978-0-321-60191-9. doi: 10.1007/s13398-014-0173-7.2.

[**23**] Oliver Hummel and Colin Atkinson. Extreme harvesting: Test driven discovery and reuse of software components. In *Proceedings of the 2004 IEEE International Conference on Information Reuse and Integration, IRI-2004*, pages 66–72, 2004. ISBN 0780388194. doi: 10.1109/IRI.2004.1431438.

[**24**] Oliver Hummel and Werner Janjic. Test-Driven Reuse: Key to Improving Precision of Search Engines for Software Reuse. In *Finding Source Code on the Web for Remix and Reuse*, pages 227–250. Springer New York, New York, NY, 2013. doi: 10.1007/978-1-4614-6596-6_12. URL `http://link.springer.com/10.1007/978-1-4614-6596-6{_}12`.

[**25**] Oliver Hummel, Werner Janjic, and Colin Atkinson. Evaluating the efficiency of retrieval methods for component repositories. In *19th International Conference on Software Engineering and Knowledge Engineering, SEKE 2007*, pages 404–409, 2007. ISBN 9781627486613 (ISBN).

[**26**] Oliver Hummel, Werner Janjic, and Colin Atkinson. Code Conjurer: Pulling Reusable Software out of Thin Air. 25(5):45–52, sep 2008. ISSN 0740-7459.

doi: 10.1109/MS.2008.110. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4602673`.

[**27**] Docker Inc. The Docker Platform. https://www.docker.com, 2017. Accessed: 19.01.2017.

[**28**] Katsuro Inoue, Reishi Yokomori, Tetsuo Yamamoto, Makoto Matsushita, and Shinji Kusumoto. Ranking significance of software components based on use relations. 31(3):213–225, 2005. ISSN 00985589. doi: 10.1109/TSE. 2005.38.

[**29**] Kenneth Pugh. *Lean-Agile Acceptance Test-Driven Development: Better Software Through Collaboration*. 2011. ISBN 0321714083. doi: 10.1145/1988997.1989006. URL `http://books.google.com/books?id=dyAa{_}gAACAAJ{&}pgis=1`.

[**30**] Marcus Kessel and Colin Atkinson. Ranking software components for reuse based on non-functional properties. pages 1–29, jul 2016. ISSN 1387-3326. doi: 10.1007/s10796-016-9685-3. URL `http://link.springer.com/10.1007/s10796-016-9685-3`.

[**31**] Y. Kim and E.A. Stohr. Software reuse: issues and research directions. In *Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences*, pages 612–623 vol.4. IEEE, 1992. ISBN 0-8186-2420-5. doi: 10. 1109/HICSS.1992.183360. URL `http://ieeexplore.ieee.org/document/183360/`.

[**32**] Charles W. Krueger and Charles W. Software reuse. 24(2):131–183, jun 1992. ISSN 03600300. doi: 10.1145/130844.130856. URL `http://portal.acm.org/citation.cfm?doid=130844.130856`.

[**33**] Maarit Laanti, Outi Salo, and Pekka Abrahamsson. Agile methods rapidly replacing traditional methods at Nokia: A survey of opinions on agile transformation. 53(3):276–290, 2011. doi: 10.1016/j.infsof.2010.11.010.

[**34**] Otávio Augusto Lazzarini Lemos, Sushil Krishna Bajracharya, and Joel Ossher. CodeGenie: a Tool for Test-Driven Source Code Search. 07pp:525–526, 2007. doi: 10.1145/1321631.1321726. URL `http://portal.acm.org/citation.cfm?id=1321631.1321726`.

[35] Marko Leppanen, Simo Makinen, Max Pagels, Veli-Pekka Eloranta, Juha Itkonen, Mika V. Mantyla, and Tomi Mannisto. The highways and country roads to continuous deployment. 32(2):64–72, mar 2015. doi: 10.1109/MS.2015.50. URL http://ieeexplore.ieee.org/document/7057604/.

[36] Matt Mackall. Towards a Better SCM: Revlog and Mercurial. *Linux Symposium*, 2:91–98, 2006. URL https://mercurial.selenic.com/wiki/Presentations?action=AttachFile{&}do=get{&}target=ols-mercurial-paper.pdf.

[37] Mika V. Mäntylä, Bram Adams, Foutse Khomh, Emelie Engström, and Kai Petersen. On rapid releases and software testing: a case study and a semi-systematic literature review. 20(5):1384–1425, oct 2015. doi: 10.1007/s10664-014-9338-4. URL http://link.springer.com/10.1007/s10664-014-9338-4.

[38] Robert C Martin. *Agile Software Development, Principles, Patterns, and Practices*. 2002. ISBN 0135974445. doi: 10.1007/BF03250842.

[39] M.D. McIlroy. Mass Produced Software Components. pages 138–155, 1968.

[40] a. Mili, R. Mili, and R.T. Mittermeir. A survey of software reuse libraries. 5(1):349–414–414, 1998. ISSN 1022-7091. doi: 10.1023/A:1018964121953. URL http://www.springerlink.com/content/u7j724230tn12h03/.

[41] Hafedh Mili. *Reuse based software engineering : techniques, organization and measurement*. Wiley, 2002. ISBN 0471398195.

[42] M. Morisio, M. Ezran, and C. Tully. Success and failure factors in software reuse. 28(4):340–357, apr 2002. ISSN 0098-5589. doi: 10.1109/TSE.2002.995420. URL http://ieeexplore.ieee.org/document/995420/.

[43] Steve Neely and Steve Stolt. Continuous Delivery? Easy! Just Change Everything (Well, Maybe It Is Not That Easy). In *2013 Agile Conference*, pages 121–128. IEEE, aug 2013. ISBN 978-0-7695-5076-3. doi: 10.1109/AGILE.2013.17. URL http://ieeexplore.ieee.org/document/6612887/.

[44] C. Michael. Pilato, Ben. Collins-Sussman, and Brian W. Fitzpatrick. *Version control with subversion*. O'Reilly, 2008. ISBN 9780596510336.

[**45**] Thomas P. Pole and W.B. Frakes. An Empirical Study of Representation Methods for Reusable Software Components. 20(8):617–630, 1994. ISSN 00985589. doi: 10.1109/32.310671.

[**46**] Mary (Mary B.) Poppendieck and Thomas David. Poppendieck. *Lean software development : an agile toolkit.* Addison-Wesley, 2003. ISBN 0321150783.

[**47**] Rubén Prieto-Díaz and Peter Freeman. Classifying Software for Reusability. 4(1):6–16, 1987. ISSN 07407459. doi: 10.1109/MS.1987.229789.

[**48**] Steven P. Reiss. Semantics-based code search. In *Proceedings - International Conference on Software Engineering*, pages 243–253, 2009. ISBN 9781424434527. doi: 10.1109/ICSE.2009.5070525.

[**49**] Kim Rejström. Implementing continuous integration in a small company: A case study. G2 pro gradu, diplomityö, 2016-10-27. URL `http://urn.fi/URN:NBN:fi:aalto-201611025461`.

[**50**] Pilar Rodríguez, Alireza Haghighatkhah, Lucy Ellen Lwakatare, Susanna Teppola, Tanja Suomalainen, Juho Eskeli, Teemu Karvonen, Pasi Kuvaja, June M. Verner, and Markku Oivo. Continuous deployment of software intensive products and services: A systematic mapping study. 123:263–291, 2016. doi: 10.1016/j.jss.2015.12.015.

[**51**] RC Seacord. Software engineering component repositories. 1999. URL `http://faculty.ksu.edu.sa/ghazy/CBD{_}MSc/Ref-25.pdf{#}page=95`.

[**52**] Perforce Software. Continuous Delivery: The New Normal for Software Development. https://www.perforce.com/pdf/continuous-delivery-report.pdf, 2015. Accessed: 21.02.2017.

[**53**] Daniel Ståhl and Jan Bosch. Automated software integration flows in industry: a multiple-case study. In *Companion Proceedings of the 36th International Conference on Software Engineering - ICSE Companion 2014*, pages 54–63, New York, New York, USA, 2014. ACM Press. ISBN 9781450327688. doi: 10.1145/2591062.2591186. URL `http://dl.acm.org/citation.cfm?doid=2591062.2591186`.

[**54**] Thomas A. Standish. An Essay on Software Reuse. SE-10(5):494–497, 1984. ISSN 00985589. doi: 10.1109/TSE.1984.5010272.

[**55**] Kathryn T. Stolee, Sebastian Elbaum, and Matthew B. Dwyer. Code search with input/output queries: Generalizing, ranking, and assessment. 116:35–48, 2016. ISSN 01641212. doi: 10.1016/j.jss.2015.04.081.

[**56**] Suresh Thummalapenta and Tao Xie. Parseweb: a programmer assistant for reusing open source code on the web. pages 204–213, 2007. ISSN 03621340. doi: 10.1145/1321631.1321663. URL `http://dl.acm.org/citation.cfm?id=1321663`.

[**57**] Kristina Weis. GitHub CEO and Co-Founder Chris Wanstrath Keynoting Esri's DevSummit! — ArcGIS Blog. https://goo.gl/or7oIX, 2014. Accessed: 14.01.2017.

Appendix

A. Simile - Code

In this section we will expose the different Java classes of our prototype.

A.1. de.unimannheim.informatik.swt.simile

```
package de.unimannheim.informatik.swt.simile;

import com.google.common.base.Strings;
import de.unimannheim.informatik.swt.simile.services.*;
import org.apache.commons.lang3.exception.ExceptionUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.scheduling.annotation.Async;
import org.springframework.stereotype.Service;

import java.io.File;
import java.io.IOException;
import java.util.List;

@Service
public class Simile {

    private final SocoraRequester socoraRequester;

    @Autowired
    public Simile(SocoraRequester socoraRequester) {
            this.socoraRequester = socoraRequester;
    }

    @Async
    public void searchForComponents(
    String repo,
    String branch,
```

```
30        String folder,
          String recipient
          ) throws IOException, InterruptedException {

          new Cloner(repo, branch, folder).cloneRepository();
35
              JavaClassFilter javaClassFilter = new JavaClassFilter();

              File maintDir = new File(String.format("%s/src/main", folder
                  ));
              JavaClassHandler javaClassHandler = new JavaClassHandler();
40            new DirectoryExplorer(javaClassHandler, javaClassFilter).
                  explore(maintDir);

              File testDir = new File(String.format("%s/src/test", folder)
                  );
              JavaClassHandler testClassHandler = new JavaClassHandler();
              new DirectoryExplorer(testClassHandler, javaClassFilter).
                  explore(testDir);
45
              this.makeRequestWithClasses(javaClassHandler.
                  getClassesMQLNotation(), recipient);
              this.makeRequestWithTestClasses(testClassHandler.
                  getTestClassesCode(), recipient);
          }

50        private void makeRequestWithClasses(List<String> classes, String
              recipientEmail) {
              classes.forEach(c -> {
                  try {
                      socoraRequester
                  .searchComponent(c, SocoraRequester.
                      INTERFACE_DRIVEN_SEARCH, recipientEmail);
55                } catch (Exception ex) {
                      logger.error(ExceptionUtils.getStackTrace(ex));
                  }
              });
          }
60
          private void makeRequestWithTestClasses(List<String> testClasses
              , String recipientEmail) {
              testClasses.forEach(tc -> {
```

```
            try {
                socoraRequester
            .searchComponent(tc, SocoraRequester.TEST_DRIVEN_SEARCH,
                recipientEmail);
            } catch (Exception ex) {
                logger.error(ExceptionUtils.getStackTrace(ex));
            }
        });
    }
}
```

<div align="center">Listing A.1: Simile.java</div>

```
package de.unimannheim.informatik.swt.simile;

import org.springframework.boot.builder.SpringApplicationBuilder;
import org.springframework.boot.web.support.
    SpringBootServletInitializer;
import org.springframework.context.annotation.Configuration;

/**
 * Implementation of SpringBootServletInitializer necessary to
     generate a deployable .war
 * file.
 */
@Configuration
public class ServletInitializer extends SpringBootServletInitializer
    {
  @Override
  protected SpringApplicationBuilder configure(
      SpringApplicationBuilder application) {
    return application.sources(SimileApplication.class);
  }
}
```

<div align="center">Listing A.2: ServletInitializer.java</div>

```
package de.unimannheim.informatik.swt.simile;

import freemarker.template.Configuration;
import freemarker.template.TemplateExceptionHandler;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

```java
    import org.springframework.boot.SpringApplication;
    import org.springframework.boot.autoconfigure.SpringBootApplication;
    import org.springframework.context.annotation.Bean;
10  import org.springframework.scheduling.annotation.
        AsyncConfigurerSupport;
    import org.springframework.scheduling.annotation.EnableAsync;
    import org.springframework.scheduling.concurrent.
        ThreadPoolTaskExecutor;

    import java.io.File;
15  import java.io.IOException;
    import java.util.concurrent.Executor;

    @SpringBootApplication
    @EnableAsync
20  public class SimileApplication extends AsyncConfigurerSupport {

        public static void main(String[] args) {
            SpringApplication.run(SimileApplication.class, args);
        }
25
        @Override
        public Executor getAsyncExecutor() {
            ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor
                ();
            executor.setCorePoolSize(2);
30          executor.setMaxPoolSize(2);
            executor.setQueueCapacity(500);
            executor.setThreadNamePrefix("SIMILE-");
            executor.initialize();
            return executor;
35      }

        @Bean
        public Configuration freeMarkerConfiguration() throws
            IOException {
            Configuration cfg = new Configuration(Configuration.
                VERSION_2_3_23);
40          cfg.setDirectoryForTemplateLoading(
            new File(getClass().getClassLoader()
                        .getResource("templates")
                        .getPath()));
```

```
            cfg . setDefaultEncoding ( "UTF−8" ) ;
45          cfg . setTemplateExceptionHandler ( TemplateExceptionHandler .
                RETHROW_HANDLER) ;
            cfg . setLogTemplateExceptions ( false ) ;
            return  cfg ;
        }
}
```

Listing A.3: SimileApplication.java

A.1.1. de.unimannheim.informatik.swt.simile.controllers

```
package de . unimannheim . informatik . swt . simile . controllers ;

import cool . graph . cuid . Cuid ;
import de . unimannheim . informatik . swt . simile . Simile ;
5   import de . unimannheim . informatik . swt . simile . model . Message ;
import org . slf4j . Logger ;
import org . slf4j . LoggerFactory ;
import org . springframework . beans . factory . annotation . Autowired ;
import org . springframework . http . ResponseEntity ;
10  import org . springframework . web . bind . annotation . RequestMapping ;
import org . springframework . web . bind . annotation . RequestMethod ;
import org . springframework . web . bind . annotation . RequestParam ;
import org . springframework . web . bind . annotation . RestController ;

15  import java . io . IOException ;
import java . util . Optional ;
import java . util . regex . Pattern ;

@RestController
20  public class EntryController {
        @Autowired
        private Simile simile ;

        @RequestMapping ( name = "/ repository" , method = RequestMethod .
            POST)
25      public ResponseEntity<?> setupRepository (
            @RequestParam ( name = "repo" ) String repo ,
            @RequestParam ( name = "branch" , defaultValue = "master" ,
                required = false ) String branch ,
            @RequestParam ( name = "email" ) String email
            ) throws IOException , InterruptedException {
```

```
30          if ( validateRequest ( repo , email ) . isPresent ( ) ) {
                return ResponseEntity . badRequest ( ) . body ( validateRequest (
                    repo , email ) . get ( ) ) ;
            } else {
                simile . searchForComponents ( repo , branch , Cuid . createCuid
                    ( ) , email ) ;
                return ResponseEntity . ok ( new Message ( " Repository set
                    successfully " , 200 ) ) ;
35          }
        }

        private Optional<Message> validateRequest ( String repo , String
            email ) {
            logger . debug ( " Validating required parameters in request " ) ;
40          StringBuilder str = new StringBuilder ( ) ;
            if ( repo . isEmpty ( ) )
                str . append ( " repo is required " ) ;
            if ( email . isEmpty ( ) )
                str . append ( " email is required " ) ;
45          if ( ! this . isValidEmail ( email ) )
                str . append ( " email is not valid " ) ;
            if ( ! this . isValidGithubWebURL ( repo ) )
                str . append ( " repo must be a valid Git Web URL ( e . g . https
                    : / / github . com / . . . ) " ) ;
            if ( ! str . toString ( ) . isEmpty ( ) )
50              return Optional . ofNullable ( new Message ( str . toString ( ) ,
                    400 ) ) ;
            return Optional . empty ( ) ;
        }

        /**
55       * Validates if a Git repository url is a valid Git Web URL.
         *
         * @param gitRepo Git web URL to be validated.
         * @return true if the git url is valid, otherwise false.
         * */
60      private boolean isValidGithubWebURL ( String gitRepo ) {
            final Pattern pattern = Pattern . compile ( " ( http ( s ) ? ) ( : ( / / ) ? )
                ( [ \\w \\ . @ \\ : / \\ - ˜ ] + ) ( \\ . git ) ? " ) ;
            return pattern . matcher ( gitRepo ) . matches ( ) ;
        }
```

```
65      /**
         * Validates if an email address is valid.
         *
         * @see <a href="http://www.regexr.com/3c0ol">http://www.regexr.
           com/3c0ol</a>
         *
70       * @param email email address to be validated.
         * @return true in case the email address is valid, otherwise
           false.
         * */
        private boolean isValidEmail(String email) {
            final Pattern pattern = Pattern.compile("[a−z0−9!#$%&'*+/=?^
               _`{|}~−]+" +
75              "(?:.[a−z0−9!#$%&'*+/=?^_`{|}~−]+)*@(?:[a−z0−9](?:[a−z0
                  −9−]*[a−z0−9])?" +
                ".)+[a−z0−9](?:[a−z0−9−]*[a−z0−9])?");
            return pattern.matcher(email).matches();
        }
    }
```

Listing A.4: EntryController.java

A.1.2. de.unimannheim.informatik.swt.simile.model

```
   package de.unimannheim.informatik.swt.simile.model;

   import lombok.Data;

5  import java.util.List;

   @Data
   public class Candidate {
     private String name;
10     private String fqName;
     private List<Metric> metrics;
     private String version;
     private String description;
   }
```

Listing A.5: Candidate.java

```
   package de.unimannheim.informatik.swt.simile.model;
```

```
import lombok.Data;
import lombok.RequiredArgsConstructor;

5

@RequiredArgsConstructor
@Data
public class Message {
  private final String message;
  private final int status;
}
```

<div align="center">Listing A.6: Message.java</div>

```
package de.unimannheim.informatik.swt.simile.model;

import lombok.Data;

5  @Data
public class Metric {
  private String title;
  private Double value;
}
```

<div align="center">Listing A.7: Metric.java</div>

A.1.3. de.unimannheim.informatik.swt.simile.services

```
package de.unimannheim.informatik.swt.simile.services;

import de.unimannheim.informatik.swt.simile.util.StreamGobbler;
import lombok.Getter;
5  import lombok.RequiredArgsConstructor;

import java.io.IOException;

@RequiredArgsConstructor
10 public class Cloner {
    @Getter
    private final String repo;
    @Getter
    private final String branch;
15    @Getter
    private final String folder;
```

```
    /**
     * Executes the command <code>git clone</code> with the values of
20   * repo, branch and folder. It clones the repository of specific
         branch (default master)
     * in a specified folder.
     * */
     public int cloneRepository() throws IOException {
         int exitVal = 0;
25       try {
             String command = this.setCommand(repo, branch, folder);
             Process p = Runtime.getRuntime().exec(command);
             StreamGobbler errorGobbler = new StreamGobbler(p.
                 getErrorStream(), "INFO");
             errorGobbler.start();
30           exitVal = p.waitFor();
             return exitVal;
         } catch (InterruptedException e) {
             e.printStackTrace();
         }
35       return exitVal;
     }


     String setCommand(String repo, String branch, String folder) {
         if (!branch.isEmpty()) {
40           return String
             .format("git clone %s --branch %s %s", repo, branch,
                 folder);
         } else {
             return String
             .format("git clone %s %s", repo, folder);
45       }
     }
   }
```

Listing A.8: Cloner.java

```
   package de.unimannheim.informatik.swt.simile.services;


   import lombok.RequiredArgsConstructor;


5  import java.io.File;


   @RequiredArgsConstructor
```

```
    public class DirectoryExplorer {
      private final JavaClassHandler fileHandler;
10    private final JavaClassFilter filter;

      public void explore(File root) {
        explore(0, "", root);
      }
15
      private void explore(int level, String path, File file) {
        if (file.isDirectory()) {
          for (File child : file.listFiles()) {
            explore(level + 1, path + "/" + child.getName(), child);
20        }
        } else {
          if (filter.interested(level, path, file)) {
            fileHandler.handle(level, path, file);
          }
25      }
      }
    }
```

Listing A.9: DirectoryExplorer.java

```
    package de.unimannheim.informatik.swt.simile.services;

    import org.slf4j.Logger;
    import org.slf4j.LoggerFactory;
5   import org.springframework.stereotype.Service;

    import javax.mail.*;
    import javax.mail.internet.InternetAddress;
    import javax.mail.internet.MimeMessage;
10  import java.util.Date;
    import java.util.Properties;

    @Service
    public class EmailSender {
15    private static final Logger logger = LoggerFactory.getLogger(
        EmailSender.class);

      /**
       * Sends email to <code>toEmail</code> with subject <code>subject
         </code>
```

```java
 *  and  with  body  <code>body</code>.
 *
 *  @param  toEmail  recipient  of  email.
 *  @param  subject  of  the  email.
 *  @param  body  of  the  email.
 **/
public void sendEmail(String toEmail, String subject, String body)
    {
  try {
    MimeMessage msg = new MimeMessage(getSession());
    msg.addHeader("Content−type", "text/HTML; charset=UTF−8");
    msg.addHeader("format", "flowed");
    msg.addHeader("Content−Transfer−Encoding", "8bit");

    msg.setFrom(new InternetAddress("no_reply@simile.de", "NoReply
        −Simile"));
    msg.setReplyTo(InternetAddress.parse("no_reply@simile.de",
        false));
    msg.setSubject(subject, "UTF−8");
    msg.setText(body, "UTF−8", "html");
    msg.setSentDate(new Date());

    msg.setRecipients(
      Message.RecipientType.TO,
      InternetAddress.parse(toEmail, false)
    );

    Transport.send(msg);
  } catch (Exception e) {
    e.printStackTrace();
  }
}

private Session getSession() {
  final String fromEmail = "some@email.com";
  final String password = "password";

  logger.debug("SSLEmail Start");
  Properties props = new Properties();
  props.put("mail.smtp.host", "smtp.someserver.com");
  props.put("mail.smtp.socketFactory.port", "465");
  props.put("mail.smtp.socketFactory.class", "javax.net.ssl.
```

```
            SSLSocketFactory");
        props.put("mail.smtp.auth", "true");
        props.put("mail.smtp.port", "465");
60      Authenticator auth = new Authenticator() {
            protected PasswordAuthentication getPasswordAuthentication() {
                return new PasswordAuthentication(fromEmail, password);
            }
        };
65      return Session.getDefaultInstance(props, auth);
    }
}
```

Listing A.10: EmailSender.java

```
package de.unimannheim.informatik.swt.simile.services;

import java.io.File;

5  public class JavaClassFilter implements Filter {
    @Override
    public boolean interested(int level, String path, File file) {
        return path.endsWith(".java");
    }
10 }
```

Listing A.11: JavaClassFilter.java

```
package de.unimannheim.informatik.swt.simile.services;

import com.github.javaparser.JavaParser;
import com.github.javaparser.ast.NodeList;
5  import com.github.javaparser.ast.body.Parameter;
import com.google.common.base.Strings;
import lombok.Getter;
import org.apache.commons.lang3.StringUtils;
import org.apache.commons.lang3.text.StrBuilder;
10 import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.File;
import java.io.IOException;
15 import java.util.*;
```

```java
public class JavaClassHandler {

    private List<String> methods = new ArrayList<>();
    @Getter
    private List<String> testClassesCode = new ArrayList<>();
    @Getter
    private Map<String, List<String>> classes = new HashMap<>();

    /**
     * Handles the current Java class and for each method it contains,
     * it transforms the method into Merobase Query Language format.
     */
    public void handle(int level, String path, File file) {
        try {
            JavaClassVisitor jcv = new JavaClassVisitor();
            jcv.visit(JavaParser.parse(file), null);
            jcv.getMethods()
                .forEach(method -> methods.add(this.getMethodMQLNotation(
                    method.getNameAsString(),
                    method.getParameters(),
                    method.getType().toString()
                )));
            classes.put(jcv.getClassObj().getNameAsString(), methods);
            methods = new ArrayList<>();
            if (jcv.getTestClasses().size() > 0)
                testClassesCode.addAll(jcv.getTestClasses());
        } catch (IOException e) {
            new RuntimeException(e);
        }
    }

    private String getMethodMQLNotation(String methodName, NodeList<
        Parameter> params, String returnType) {
        List<String> paramTypes = new ArrayList<>();
        params.forEach(param -> paramTypes.add(param.getType().toString
            ()));
        return String.format(
            "%s(%s):%s",
            methodName,
            StringUtils.join(paramTypes, ','),
            returnType);
    }
```

```
     /**
      * Returns  the  classes  and  the  methods  it  cotains  in  MQL  notation.
60    *
      * <p>For  example  for  the  following  class:
      * <pre>
      * public  class  TestClass {
      *   public  String  methodOne(int  param){
65    *   }
      * }
      * </pre>
      * <p>  This  method  will  return:
      * <pre>
70    *  TestClass (methodOne (int ): String ;)
      * </pre>
      * */
     public List<String> getClassesMQLNotation () {
       List<String> classesMQLNotation = new ArrayList <>();
75     if ( classes . size () > 0) {
         classes . forEach ((c, ms) -> {
           StrBuilder strBuilder = new StrBuilder ();
           strBuilder . append ( String . format ("%s(", c));
           strBuilder . append ( StringUtils . join (ms, ';'));
80         strBuilder . append ( String . format (";)"));
           logger . debug ( strBuilder . toString ());
           classesMQLNotation . add ( strBuilder . toString ());
         });
         return classesMQLNotation ;
85     }
       return Collections . emptyList ();
     }
   }
```

Listing A.12: JavaClassHandler.java

```
   package de.unimannheim . informatik . swt . simile . services ;

   import com . github . javaparser . ast . body . ClassOrInterfaceDeclaration ;
   import com . github . javaparser . ast . body . MethodDeclaration ;
5  import com . github . javaparser . ast . visitor . VoidVisitorAdapter ;
   import lombok . Getter ;
   import org . slf4j . Logger ;
   import org . slf4j . LoggerFactory ;
```

```java
10   import java.util.ArrayList;
     import java.util.List;

     public class JavaClassVisitor extends VoidVisitorAdapter {

15     @Getter
       private ClassOrInterfaceDeclaration classObj;
       @Getter
       private List<MethodDeclaration> methods = new ArrayList<>();
       @Getter
20     private List<String> testClasses = new ArrayList<>();

       /**
        * If the node visited is a Java class, it checks if it contains
        * annotation <code>@Test</code>. If true, it is added to test
           class
25      * list.
        * */
       @Override
       public void visit(ClassOrInterfaceDeclaration n, Object arg) {
         super.visit(n, arg);
30       this.classObj = n;
         if(n.toString().contains("@Test")) {
           testClasses.add(n.toString());
         }
       }
35
       /**
        * If the node visited is a method, it is added into <code>methods
           </code>
        * list.
        * */
40     @Override
       public void visit(MethodDeclaration n, Object arg) {
         super.visit(n, arg);
         methods.add(n);
       }
45   }
```

Listing A.13: JavaClassVisitor.java

```java
package de.unimannheim.informatik.swt.simile.services;
```

```
import com.github.javaparser.ast.NodeList;
import com.github.javaparser.ast.body.MethodDeclaration;
import com.github.javaparser.ast.body.Parameter;
import com.github.javaparser.ast.visitor.VoidVisitorAdapter;
import lombok.Getter;
import org.apache.commons.lang3.StringUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.ArrayList;
import java.util.List;

public class JavaMethodVisitor extends VoidVisitorAdapter {

    private static final Logger logger = LoggerFactory.getLogger(
        JavaMethodVisitor.class);

    @Getter
    private String name;
    @Getter
    private String returnType;
    @Getter
    private String parameterTypes;
    private List<String> params = new ArrayList<>();
    @Getter
    private NodeList<Parameter> parameters;

    @Override
    public void visit(MethodDeclaration n, Object arg) {
        super.visit(n, arg);
        this.name = n.getName().toString();
        this.returnType = n.getType().toString();
        n.getParameters().forEach(param -> params.add(param.getType().
            toString()));
        parameterTypes = StringUtils.join(params, ',');
        parameters = n.getParameters();
    }
}
```

Listing A.14: JavaMethodVisitor.java

```
package de.unimannheim.informatik.swt.simile.services;
```

```java
import com.github.javaparser.ast.Node;

public class NodeIterator {
    public interface NodeHandler {
        boolean handle(Node node);
    }

    private NodeHandler nodeHandler;

    public NodeIterator(NodeHandler nodeHandler) {
        this.nodeHandler = nodeHandler;
    }

    public void explore(Node node) {
        if (nodeHandler.handle(node)) {
            for (Node child : node.getChildNodes()) {
                explore(child);
            }
        }
    }
}
```

Listing A.15: NodeIterator.java

```java
package de.unimannheim.informatik.swt.simile.services;

import com.merobase.socora.engine.index.repository.candidate.
    CandidateDocument;
import com.merobase.socora.engine.index.repository.candidate.
    CandidateListResult;
import com.merobase.socora.engine.search.*;
import com.merobase.socora.engine.search.filter.Filters;
import com.merobase.socora.engine.search.ranking.Rankings;
import de.unimannheim.informatik.swt.simile.model.Candidate;
import de.unimannheim.informatik.swt.simile.model.Metric;
import freemarker.template.Configuration;
import freemarker.template.Template;
import freemarker.template.TemplateException;
import org.apache.commons.lang3.StringUtils;
import org.apache.commons.lang3.Validate;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

```
     import org.springframework.beans.factory.annotation.Autowired;
     import org.springframework.stereotype.Service;

20   import java.io.IOException;
     import java.io.StringWriter;
     import java.math.BigDecimal;
     import java.math.RoundingMode;
     import java.util.*;
25
     @Service
     public class SocoraRequester {
       public static String TEST_DRIVEN_SEARCH = "TEST_DRIVEN_SEARCH";
       public static String INTERFACE_DRIVEN_SEARCH = "
          INTERFACE_DRIVEN_SEARCH";
30     public static String KEYWORD_SEARCH = "KEYWORD_SEARCH";

       private static String baseURI = "http://socora.merobase.com/socora
          ";

       private static String user = "socora";
35     private static String pass = "d3fudj983r223dhs23";

       private final EmailSender emailSender;
       private final Configuration freeMarker;

40     @Autowired
       public SocoraRequester(EmailSender emailSender, Configuration
          freeMarker) {
         this.emailSender = emailSender;
         this.freeMarker = freeMarker;
       }
45
       public void searchComponent(
         String query,
         String searchType,
         String recipient) throws IOException, InterruptedException,
            TemplateException {
50
         Validate.notBlank(query, "Query parameter is required and cannot
             be blank");
         if (StringUtils.compare(searchType, INTERFACE_DRIVEN_SEARCH) ==
            0 ||
```

```
            StringUtils.compare(searchType, KEYWORD_SEARCH) == 0 ||
            StringUtils.isBlank(searchType)) {
55          this.textualSearchComponent(query, recipient);
        } else if (StringUtils.compare(searchType, TEST_DRIVEN_SEARCH)
            == 0) {
            this.testDrivenSearchComponent(query, recipient);
        } else {
            this.textualSearchComponent(query, recipient);
60        }
        }


    /**
     * In charge of sending the test class to SOCORA to search for
         components.
65   *
     * @param testClassSourceQuery which is sent to SOCORA to search
         for components.
     * @param recipient (email) where the result will be sent.
     * */
    private void testDrivenSearchComponent(
70      String testClassSourceQuery,
        String recipient) throws Exception {

        CandidateSearchClient candidateSearchClient =
            new CandidateSearchClient(baseURI, auth(user, pass));
75
        CandidateSearchRequest request = new CandidateSearchRequest();

        int maxResults = 400;

80      request.setInput(testClassSourceQuery);
        QueryParams queryParams = new QueryParams();
        queryParams.setRows(maxResults);

        queryParams.setArtifactInformation(true);
85      queryParams.setContent(true);

        // FILTERS
        queryParams.setFilters(Arrays.asList(
            Filters.HASH_CODE_CLONE_FILTER,
90          Filters.NO_ABSTRACT_CLASS_FILTER,
            Filters.NO_INTERFACE_FILTER,
```

```
           Filters.LATEST_VERSION_FILTER,
           Filters.FUNCTIONAL_SUFFICIENCY_FILTER
       ));

95
       request.setQueryParams(queryParams);

       request.setTestDrivenSearch(Boolean.TRUE);

100    queryParams.setRankingStrategy(Rankings.
           HYBRID_NON_DOMINATED_SORTING);

       RankingCriterion fsCriterion = new RankingCriterion();
       fsCriterion.setName("sf_instruction_leanness");
       fsCriterion.setObjective(RankingCriterion.MAX);
105    RankingCriterion performanceCriterion = new RankingCriterion();
       performanceCriterion.setName("jmh_thrpt_score_mean");
       performanceCriterion.setObjective(RankingCriterion.MAX);
       RankingCriterion lcomCriterion = new RankingCriterion();
       lcomCriterion.setName("entryClass_ckjm_ext_lcom3");
110    lcomCriterion.setObjective(RankingCriterion.MIN);
       RankingCriterion couplingCriterion = new RankingCriterion();
       couplingCriterion.setName("entryClass_ckjm_ext_ce");
       couplingCriterion.setObjective(RankingCriterion.MIN);

115    queryParams.setRankingCriteria(Arrays.asList(
           fsCriterion,
           performanceCriterion,
           lcomCriterion,
           couplingCriterion
120    ));

       CandidateSearchResponse response = candidateSearchClient.search(
           request);

       String jobId = response.getJobId();
125
       JobStatus jobStatus = null;
       while (jobStatus == null ||
           jobStatus.getStatusType() != JobStatusType.FINISHED) {

130        jobStatus = candidateSearchClient.getJobStatus(jobId);
           Thread.sleep(30 * 1000L);
```

```
          }

          if(jobStatus.getStatusType() == JobStatusType.FINISHED)
135         getTestDrivenSearchResult(jobId, recipient,
              testClassSourceQuery);
        }


      /**
       * In charge of fetching the result of a job with given id,
140      * handle it and send the result to the recipient.
       *
       * @param jobId of the job to fetch the result.
       * @param recipient (email) to where we will send the result.
       * */
145   private void getTestDrivenSearchResult(
        String jobId,
        String recipient,
        String testClassQueried) throws IOException, TemplateException {

150     CandidateSearchClient candidateSearchClient =
          new CandidateSearchClient(baseURI, auth(user, pass));

        QueryParams queryParams = new QueryParams();
        queryParams.setRows(2000);
155     queryParams.setArtifactInformation(true);
        queryParams.setContent(true);


        // FILTERS
        queryParams.setFilters(Arrays.asList(
160       Filters.HASH_CODE_CLONE_FILTER,
          Filters.NO_ABSTRACT_CLASS_FILTER,
          Filters.NO_INTERFACE_FILTER,
          Filters.LATEST_VERSION_FILTER,
          Filters.FUNCTIONAL_SUFFICIENCY_FILTER
165     ));

        queryParams.setRankingStrategy(Rankings.
          HYBRID_NON_DOMINATED_SORTING);

        RankingCriterion fsCriterion = new RankingCriterion();
170     fsCriterion.setName("sf_instruction_leanness");
        fsCriterion.setObjective(RankingCriterion.MAX);
```

```
      RankingCriterion performanceCriterion = new RankingCriterion();
      performanceCriterion.setName("jmh_thrpt_score_mean");
      performanceCriterion.setObjective(RankingCriterion.MAX);
175   RankingCriterion lcomCriterion = new RankingCriterion();
      lcomCriterion.setName("entryClass_ckjm_ext_lcom3");
      lcomCriterion.setObjective(RankingCriterion.MIN);
      RankingCriterion couplingCriterion = new RankingCriterion();
      couplingCriterion.setName("entryClass_ckjm_ext_ce");
180   couplingCriterion.setObjective(RankingCriterion.MIN);

      queryParams.setRankingCriteria(Arrays.asList(
        fsCriterion,
        performanceCriterion,
185     lcomCriterion,
        couplingCriterion
      ));

      CandidateSearchResponse response =
190     candidateSearchClient.getResults(jobId, queryParams);

      CandidateListResult result = response.getCandidates();

      this.sendResult(
195     this.processTemplateTestDrivenSearchResult(result, queryParams
          , jobId),
        recipient,
        "Test−driven search"
      );
    }
200
    /**
     * In charge of making the request to SOCORA using textual search.
     * The result is sent to the recipient.
     *
205  * @param method in MQL format that will be sent to SOCORA.
     * @param recipient (email) to which the result will be sent.
     * */
    private void textualSearchComponent(
      String method,
210   String recipient) throws IOException, TemplateException {

      CandidateSearchClient candidateSearchClient =
```

```
        new CandidateSearchClient(baseURI, auth(user, pass));

215     CandidateSearchRequest request = new CandidateSearchRequest();

        int maxResults = 10;

        request.setInput(method);
220     QueryParams queryParams = new QueryParams();
        queryParams.setRows(maxResults);

        queryParams.setArtifactInformation(true);
        queryParams.setContent(true);
225
        // FILTERS
        queryParams.setFilters(Arrays.asList(
          Filters.HASH_CODE_CLONE_FILTER,
          Filters.NO_ABSTRACT_CLASS_FILTER,
230       Filters.NO_INTERFACE_FILTER,
          Filters.LATEST_VERSION_FILTER,
          Filters.FUNCTIONAL_SUFFICIENCY_FILTER
        ));

235     request.setQueryParams(queryParams);
        request.setTestDrivenSearch(Boolean.FALSE);

        queryParams.setRankingStrategy(Rankings.SINGLE_OBJECTIVE);

240     List<RankingCriterion> rankingCriteria = Collections.
            singletonList(
          RankingCriterionBuilder
            .rankingCriterion()
            .withName("luceneScore")
            .withObjective(RankingCriterion.MAX)
245         .build()
        );

        queryParams.setRankingCriteria(rankingCriteria);

250     CandidateSearchResponse response = candidateSearchClient.search(
            request);

        CandidateListResult result = response.getCandidates();
```

```
            this.sendResult(
255             this.processTemplateTextualSearchResult(method, result,
                    queryParams),
                recipient,
                "Textual search"
            );
        }
260

        /**
         * Processes the template using the candidates resulted from
         * test-driven search.
         *
265      * @param result from interface-driven search.
         * @param queryParams to show more data.
         * @param jobId
         *
         * @return result in a human-readable format.
270      * */
        private String processTemplateTestDrivenSearchResult(
            CandidateListResult result,
            QueryParams queryParams,
            String jobId) throws IOException, TemplateException {

275
            Template emailTmp = freeMarker.getTemplate("
                testdrivenResultEmail.ftl");
            StringWriter stringWriter = new StringWriter();
            List<Candidate> candidates = this.extractSearchCandidates(result
                , queryParams);
            Map<String, Object> root = new HashMap<>();
280         root.put("jobId", jobId);
            root.put("numMetrics", candidates.get(0).getMetrics().size());
            root.put("metrics", candidates.get(0).getMetrics());
            root.put("candidates", candidates);
            emailTmp.process(root, stringWriter);
285
            return stringWriter.toString();
        }


        /**
290      * Processes the template using the candidates resulted from
         * interface-driven search.
```

```
         *
         * @param methodQueried
         * @param result from interface−driven search.
295      * @param queryParams to show more data.
         *
         * @return result in a human−readable format.
         * */
        private String processTemplateTextualSearchResult(
300         String methodQueried,
            CandidateListResult result,
            QueryParams queryParams) throws IOException, TemplateException {

            Template emailTmp = freeMarker.getTemplate("interfaceResultEmail
                .ftl");
305         StringWriter stringWriter = new StringWriter();
            List<Candidate> candidates =
                this.extractSearchCandidates(result, queryParams);
            Map<String, Object> root = new HashMap<>();
            root.put("query", methodQueried);
310         root.put("numMetrics", candidates.get(0).getMetrics().size());
            root.put("metrics", candidates.get(0).getMetrics());
            root.put("candidates", candidates);
            emailTmp.process(root, stringWriter);

315         return stringWriter.toString();
        }


        /**
         * Transforms result into a human−readable format.
320      * */
        private List<Candidate> extractSearchCandidates(
            CandidateListResult result,
            QueryParams queryParams) {

325         List<Candidate> candidates = new ArrayList<>();
            result.getCandidates().forEach(doc −> {
              Candidate candidate = new Candidate();
              candidate.setName(
                Optional.ofNullable(doc.getArtifactInfo().getName()).orElse(
                  ""));
330           candidate.setFqName(
                Optional.ofNullable(doc.getFQName()).orElse(""));
```

```
          candidate . setMetrics (
            extractMetrics ( doc . getMetrics ( ) , queryParams .
              getRankingCriteria ( ) ) ) ;
          candidate . setDescription (
335         Optional . ofNullable ( doc . getArtifactInfo ( ) . getDescription ( ) ) .
              orElse ( "" ) ) ;
          candidate . setVersion (
            Optional . ofNullable ( doc . getVersion ( ) ) . orElse ( "" ) ) ;
          candidates . add ( candidate ) ;
        } ) ;
340       return candidates ;
      }


      /**
       * Sends  the  result  of  a  search  to  the  email .
345      *
       * @param  result  which  will  be  sent .
       * @param  email  to  which  the  result  will  be  sent .
       * */
      private void sendResult ( String  result , String  email , String
          searchType ) {
350       emailSender . sendEmail (
            email ,
            String . format ( "Feedback  from  Simile  − %s" , searchType ) , result
              ) ;
      }


355   /**
       * Return  rank  for  passed  { @link  Map } based  on  given
       * CandidateRankingStrategy
       *
       * @param  ranking
360      *             { @link  Map } instance
       * @param  candidateRankingStrategy
       *             CandidateRankingStrategy  ID  { @link  Rankings }
       * @param  partialOrder
       *             true  if  partial  order  should  be  returned  ( disables
          strict
365      *             order ! ) . Candidates  in  non−distinguishable  sets
          possess  the
       *             same  ranks .
       * @return  rank  for  given  { @link  } based  on  passed
```

```java
 *              CandidateRankingStrategy
 */
private static int getRank(
  Map<String, Double> ranking,
  String candidateRankingStrategy,
  boolean partialOrder) {

  Validate.notBlank(
    candidateRankingStrategy,
    "CandidateRankingStrategy ID cannot be blank");

  String key = candidateRankingStrategy;
  if (partialOrder) {
    key += "_po";
  }

  return ranking.get(key).intValue();
}

private static Auth auth(String user, String pass) {
  Auth auth = new Auth();
  auth.setUser(user);
  auth.setPassword(pass);

  return auth;
}

private static List<Metric> extractMetrics(
  Map<String, Double> metrics,
  List<RankingCriterion> criteria) {

  List<Metric> metricList = new ArrayList<>();
  for (String key : metrics.keySet()) {
    for (RankingCriterion criterion : criteria) {
      if (StringUtils.equals(criterion.getName(), key)) {
        Metric m = new Metric();
        m.setTitle(SocoraRequester.getMetricTitle(key));
        m.setValue(SocoraRequester.round(metrics.get(key), 2));
        metricList.add(m);
      }
    }
  }
```

The line numbers shown in the left margin are 370, 375, 380, 385, 390, 395, 400, 405.

```
410      return metricList;
      }

      private static String getMetricTitle(String metricKey) {
        String metricTitle;
415      switch (metricKey) {
          case "luceneScore":
            metricTitle = "Lucene Score";
            break;
          case "sf_instruction_leanness":
420          metricTitle = "Leanness default (Instruction)";
            break;
          case "jmh_thrpt_score_mean":
            metricTitle = "Throughput, operations / second";
            break;
425      case "entryClass_ckjm_ext_lcom3":
            metricTitle = "Cohesion LCOM3 (entry class)";
            break;
          case "entryClass_ckjm_ext_ce":
            metricTitle = "Efferent Coupling (entry class)";
430          break;
          default:
            throw new IllegalArgumentException(
              String.format("Metric %s is not valid", metricKey));
        }
435      return metricTitle;
      }

      private static double round(double value, int places) {
        if (places < 0) throw new IllegalArgumentException();
440
        BigDecimal bd = new BigDecimal(value);
        bd = bd.setScale(places, RoundingMode.HALF_UP);
        return bd.doubleValue();
      }
445
      /**
       * Sort {@link CandidateDocument} by rank.
       *
       * @author Marcus Kessel
450      *
       */
```

```java
    private static class SortByRank implements Comparator<
        CandidateDocument> {

      private final String candidateRankingStrategy;
455   private final boolean partialOrder;

      private SortByRank(String candidateRankingStrategy, boolean
          partialOrder) {
        Validate.notBlank(candidateRankingStrategy);
        this.candidateRankingStrategy = candidateRankingStrategy;
460     this.partialOrder = partialOrder;
      }

      @Override
      public int compare(CandidateDocument o1, CandidateDocument o2) {
465     int rank1 = getRank(
          o1.getRanking(),
          candidateRankingStrategy,
          partialOrder);
        int rank2 = getRank(
470       o2.getRanking(),
          candidateRankingStrategy,
          partialOrder);

        return Comparator.<Integer>naturalOrder().compare(rank1, rank2
          );
475   }

    }

  }
```

Listing A.16: SocoraRequester.java

A.1.4. de.unimannheim.informatik.swt.simile.util

```java
package de.unimannheim.informatik.swt.simile.util;

import java.io.BufferedReader;
import java.io.IOException;
5 import java.io.InputStream;
import java.io.InputStreamReader;
```

```java
public class StreamGobbler extends Thread {
    InputStream is;
10  String type;

    public StreamGobbler(InputStream is, String type) {
        this.is = is;
        this.type = type;
15  }

    public void run() {
        try {
            InputStreamReader isr = new InputStreamReader(is);
20          BufferedReader br = new BufferedReader(isr);
            String line = null;
            while ((line = br.readLine()) != null)
                System.out.println(type + ">" + line);
        } catch (IOException ioe) {
25          ioe.printStackTrace();
        }
    }
}
```

Listing A.17: StreamGobbler.java

B. Installation of Simile

In this chapter we will show how to install the tools we used for implementing Simile. First we will start installing the CI Server Jenkins and explain how to configure it. Then, we will continue with installing Simile Jenkins plugin. After that we will explain how to configure Simile Jenkins plugin using the test project.

B.1. Installation of Jenkins

In this section we will explain how to install Jenkins CI server using Docker and Tomcat 9.

B.1.1. Docker

For installing Jenkins, we will use Docker containers and in this subsection we will explain how to install it in different platform such as macOS and Linux.

B.1.1.1. Docker on macOS

For macOS we will use Docker for Mac tool. This is an integrated, easy-to-deploy environment for building, assembling, and shipping applications from a Mac. Moreover, it is a native Mac application architected from scratch, with a native user interface and auto-update capability, deeply integrated with OS X native virtualization, Hypervisor Framework, networking and file system, making it faster and more reliable than previous ways of getting Docker on a Mac [27].

First we will download the tool from the following link: `https://download.docker.com/mac/stable/Docker.dmg`.

After downloading the *dmg* image, double-click on it and you will see something like figure B.1. When you get that image, drag and drop the Docker icon to the

Applications folder. After that, go to Applications folder and double click on Docker application.
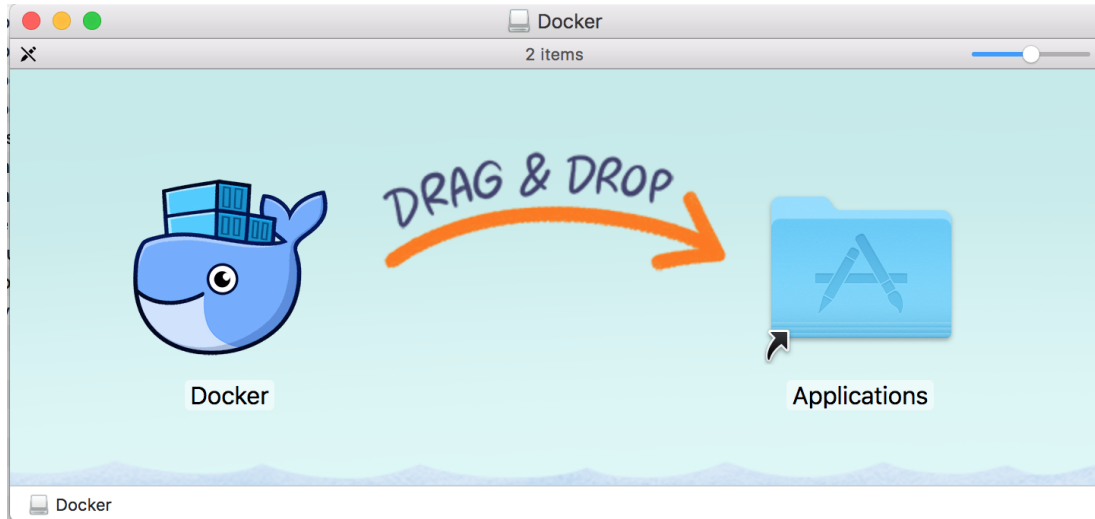

Figure B.1.: Docker for Mac installation

B.1.1.2. Docker on Linux

In this section we will explain how to install Docker in Linux, specifically in Debian (Jessie). For other distros please refer to Docker documentation[1].

First of all we need to update the repositories.

```
1 $ sudo apt-get udpate
```

Then we need to install the packages to allow *apt* to use repositories iver HTTPS.

```
1 $ sudo apt-get install apt-transport-https \
2                        ca-certificates \
3                        software-properties-common
```

Then we add the official Docker's GPG key.

---

[1]https://docs.docker.com/engine/installation/linux/ (accessed: 21.01.2017)

```
1 $ curl -fsSL https://yum.dockerproject.org/gpg | sudo apt-key add -
```

Finally we add the stable repository of Docker and update repositories.

```
1 $ sudo add-apt-repository \
2        "deb https://apt.dockerproject.org/repo/ \
3        debian-$(lsb_release -cs) \
4        main"
5 $ sudo apt-get update
```

Once we added the new repositories, we update the repository list.

```
1 $ sudo apt-get update
```

Then we install docker.

```
1 $ sudo apt-get -y install docker-engine
```

After the installation is done we can make a little test to be sure that everything was installed correctly. The following command will download a test image and runs it in a container. Once it is running will print an informational message and exits.

```
1 $ sudo docker run hello-world
```

B.1.2. Jenkins in Docker

Open a terminal on your mac and enter the following command to download the Jenkins image for Docker.

```
1 $ docker pull jenkins
2   Using default tag: latest
```

```
3   latest: Pulling from library/jenkins
4
5   # some irrelevant output was remove
6   Digest: sha256:5046434030be395ec977c98e11...
7   Status: Downloaded newer image for jenkins:latest
```

Then, we just need to run the Jenkins image with the following command.

```
1  $ docker run -p 8080:8080 -p 50000:50000 jenkins
2     # some irrelevant output was remove
3     ************************************************************
4
5     Jenkins initial setup is required. An admin user has been
6     created and a password generated.
7     Please use the following password to proceed to installation:
8
9     95199411f1894bfa97e937147c41aa62 # IMPORTANT, default admin pass
10
11    This may also be found at: /var/jenkins_home/secrets/initial.
12
13    ************************************************************
14    # some irrelevant output was removed
15    Jan 19, 2017 8:28:05 AM hudson.model.AsyncPeriodicWork$1 run
16    INFO: Finished Download metadata. 25,312 ms
```

After that, open the following URL `http://localhost:8080` and you should see some like figure B.2.
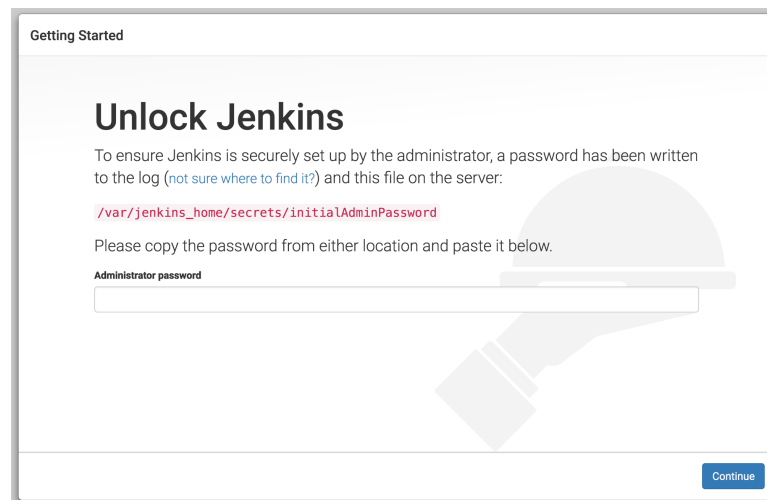
Figure B.2.: First page of Jenkins setup

There enter the password generated by Jenkins and that appeared in the logs, and then click on continue.

In the folowing page click on *Install suggested plugins* like the figure B.3.
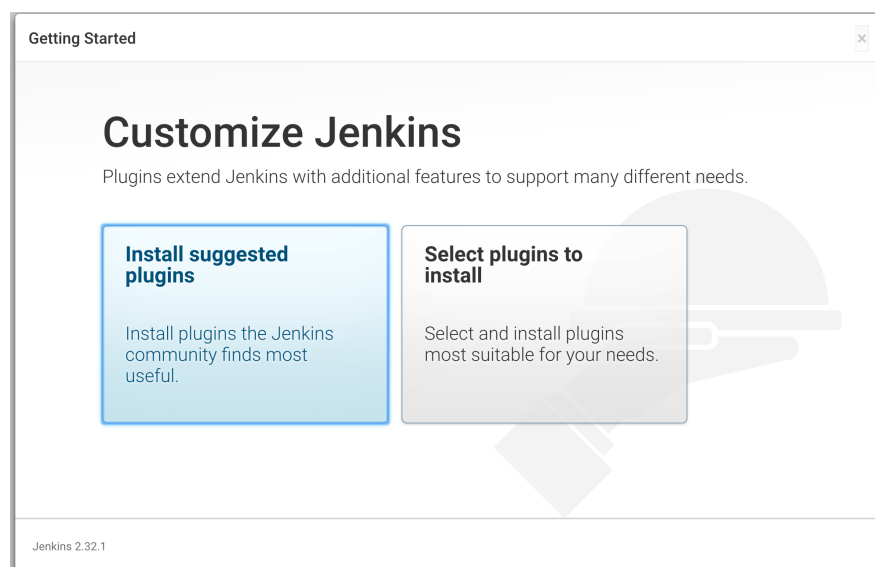

Figure B.3.: Second page of Jenkins setup

Then you should see something like the figure B.4. Here we just need to wait untill the plugins installation finish.
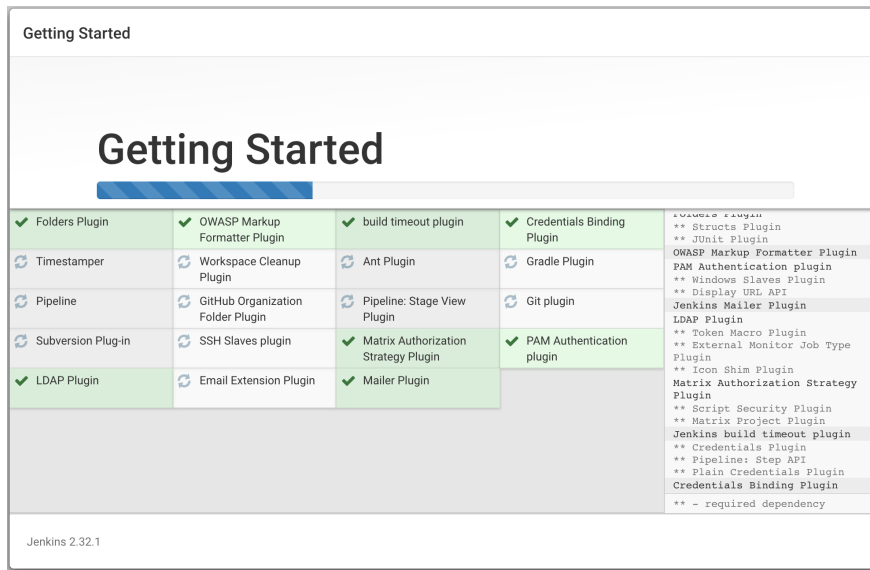
Figure B.4.: Third page of Jenkins setup

After the installation of plugins is done, click on *Continue as admin* or create custom admin user. Then you should see the home page of Jenkins (figure)
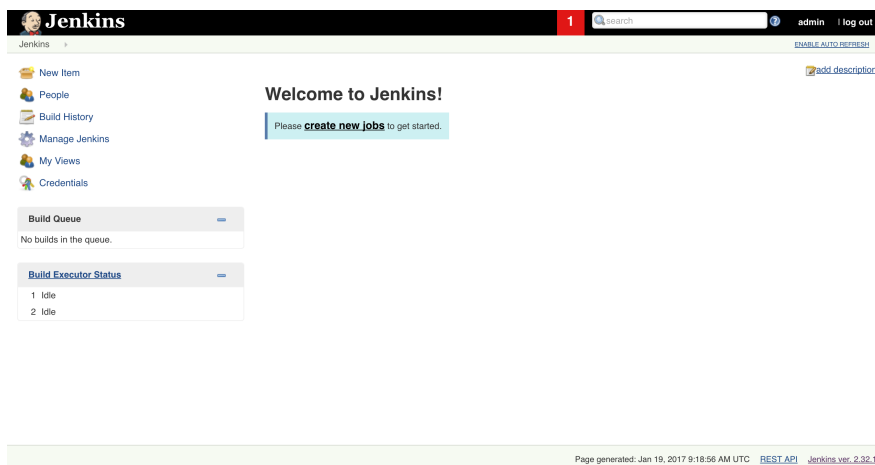

Figure B.5.: Jenkins home page

B.1.3. Jenkins in Tomcat 9

NOTE: The following steps are based on a UNIX system.

First, go to the following URL and download Tomcat 9: `http://tomcat.apache.org/download-90.cgi`.

Once downloaded, unzip the file and modify the file *tomcat-users.xml* which is under the folder *conf* of where Tomcat was unzipped. Add the following two lines within ¡tomcat-users¿¡/tomcat-users¿ to setup an admin user.

```
1 <role rolename="manager-gui"/>
2 <user username="admin" password="1234" roles="manager-gui"/>
```

Now open a terminal and enter the following command to run Tomcat server.

```
1 $ cd folder/of/tomcat
2 $ cd bin
3 $ sh catalina.sh start
```

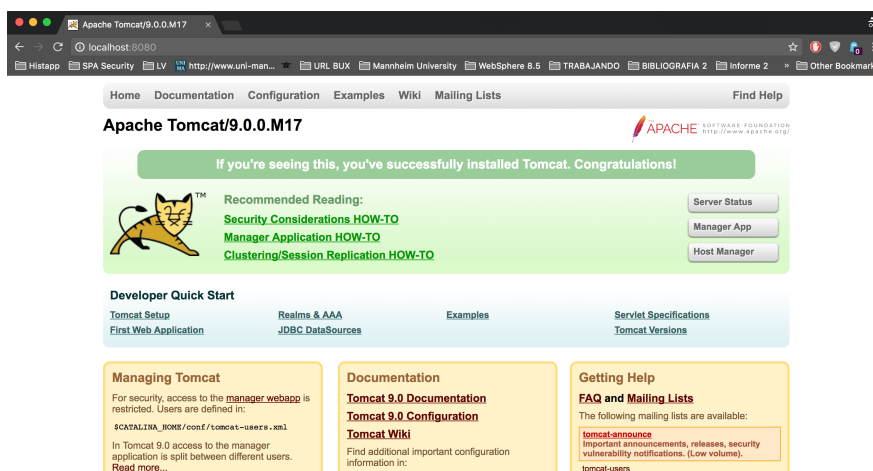Then open the following URL and you should see something like Figure B.6: `http://localhost:8080`

Figure B.6.: Tomcat home page

To install Jenkins, we first need to download the *.war* file. To do so go to the following URL: `http://mirrors.jenkins.io/war-stable/latest/jenkins.war`.

Once downloaded, move the file into the folder *webapps* which is into the folder where tomcat was installed. Then enter the following commands to restart tomcat.

```
1 $ sh catalina.sh stop
2 $ sh catalina.sh start
```

If you go to `localhost:8080/jenkins` you should see the same page like Figure B.2.

B.2. Simile Jenkins Plugin installation

To install Simile Jenkins plugin, we need to follow the following steps.

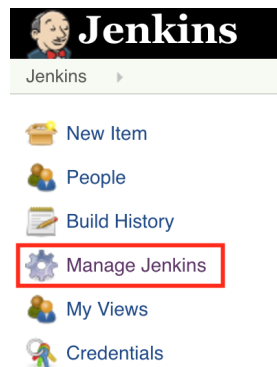First, open Jenkins URL. Then click on *Manage Jenkins* (figure B.7).

Figure B.7.: Manage Jenkins option

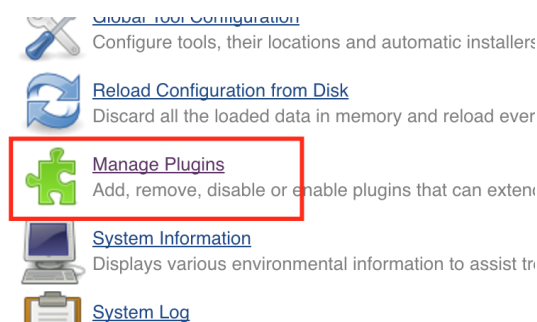Then click on *Manage Plugins* (figure B.8).

Figure B.8.: Manage Plugins option

Then go to *Advanced*, scroll down until *Upload Plugin* section (figure B.9).

Upload Plugin

You can upload a .hpi file to install a plugin from outside the central plugin repository.

File: Choose File No file chosen

Upload

Figure B.9.: Upload Plugin section

There click on choose file and select the hpi file *simile-jenkins-plugin.hpi* provided with the CD.

B.3. Simile in Tomcat 9

To install Simile in Tomcat 9, we just need to copy the file *simile.war*, provided with the CD, into the folder *webapps* which into the folder where Tomcat was installed. Once done, we just need to restart tomcat.

Eidesstattliche Erklärung

Hiermit versichere ich, dass diese Abschlussarbeit von mir persönlich verfasst ist und dass ich keinerlei fremde Hilfe in Anspruch genommen habe. Ebenso versichere ich, dass diese Arbeit oder Teile daraus weder von mir selbst noch von anderen als Leistungsnachweise andernorts eingereicht wurden. Wörtliche oder sinngemäße Übernahmen aus anderen Schriften und Veröffentlichungen in gedruckter oder elektronischer Form sind gekennzeichnet. Sämtliche Sekundärliteratur und sonstige Quellen sind nachgewiesen und in der Bibliographie aufgeführt. Das Gleiche gilt für graphische Darstellungen und Bilder sowie für alle Internet-Quellen.

Ich bin ferner damit einverstanden, dass meine Arbeit zum Zwecke eines Plagiatsabgleichs in elektronischer Form anonymisiert versendet und gespeichert werden kann. Mir ist bekannt, dass von der Korrektur der Arbeit abgesehen werden kann, wenn die Erklärung nicht erteilt wird.

Ort, den Datum

Piero Antonio Divasto Martínez