Leveraging Continuous Integration process to improve Software Reuse

Master Thesis

submitted: January 2017

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

Student ID Number: 012345

University of Mannheim
Chair of Software Engineering
B6, 26, Gebäudeteil C
D – 68159 Mannheim
Phone: +49 621-181-1691, Fax +49 621-181-1692
Internet: http://swt.informatik.uni-mannheim.de/

Contents

List of Figures

List of Tables

1. Introduction

2. Foundations

2.1. Software Reuse

Douglas McIlroy [32] 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. Although the IT industry has tried for many years to improve the speed and reduce cots of software development by reusing components, the McIlroy's vision is still the exception rather than the rule.

Although the definition of software reuse seems trivial, it is possible to find several definitions in the literature. However, most of them are similar to the definition proposed by [27]:

"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 [14] defined a list 2.1 of potentially reusable software artifacts:

However, reuse traditionally means the reuse of code fragments and components [34]. When we talk about components, we mean any cohesive and compact unit of software functionality with a well defined interface [22]. Therefore a component can be simple classes or even web-services or Enterprise Beans.

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 [14].

Software reuse emerges as a solution for the so called software development crisis [26]. 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 [13].

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 [1]. 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 strategy 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 [1].

2.1.2. Challenges of Software Reuse

Although software reuse brings many benefits, it has failed to take off. Find the right third-party software component to be reused based on a well-defined specification is one of the most challenging approaches because it requires a clear-cut matching of the potential reuse candidate and the given specification. Although there are several search tools available, most of them are still text-based and it neither reflect nor support the need to match the reuse candidates with the syntactic and semantic characteristics of a specification [20]. [20], based on software retrieval literature, identified fours problems for implementing a sustainable reuse repository.

- Repository problem [43]: This reuse repository should create and maintain a big enough software collection to provide promise search.
- Representation problem [37]: The repository should represent and index its content in a way that makes it easily accessible.
- Usability problem [16]: The repository should permit characterizing a desired component with reasonable effort and precision.
- Retrieval problem [39]: The repository should execute the queries with high precision to retrieve the desired component.

Although the two first challenges have been address in 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 swathes of free software, thus the problem of sources of components is not problem 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 problem can be regarded as solved [20]. For the second problem, clever ranking approaches such as the ranking proposed by [24] 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.

Beside the technical problems described above, organizational challenges and human factors have been identified as potential inhibitors to a successful implementation of reuse practices [35]. Business strategy, management commitment, and company culture are organizational factors that might affect software reuse [45]. Moreover, technical problems are strongly related to human factors, such as cognitive efforts, program understanding, and motivation. In the work of [1] 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 which 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.
- Measurement:
- 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.

2.1.3. Code-search engines

Code-search engines are the heart of the new generations of reuse support tools. For example, Code Conjurer [22] uses Merobase component search engine, CodeGenie relies on Sourcerer [29] or ParseWeb that works over Google Code Search [1] [47]. Next we will describe some code search engines that are still online the time of this work.

- searchcode[2] searchcode 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 shouldn't need to click through, but can if required.
- NerdyData[3] It is a search engine for source code. It supports HTML, Javascript and CSS. With this search engine you can retrieve the web

[1]Shutdown January, 15 2012 [17]
[2]https://searchcode.com/ (accessed: 13.01.2017)
[3]https://nerdydata.com (accessed: 13.01.2017)

pages which are using a specific file. For example if you look up for *font-awesome.min.css* it will retrieve the sites that are using this file.

- Spars-J[4] It 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: It 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 [46].
- Merobase[6] Merobase 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. 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[21].

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

[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 [8].

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 described above are invoked and try to extract as much syntactic information from the query as possible. If none of the parsers recognizes 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 [20].

The current implementation of Merobase, which will we used in our prototype, has been enhanced in several way since the implementation described above [25]. 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, 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 2.1.2, several problems of software reuse have been already addressed. Now one of the big problems is how to choose the so-called best component. In this section we will talk about the different component retrieval tech-

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

niques proposed in the literature. Then we will explain further what test-driven component retrieval means.

[33] divided the component retrieval techniques in six independent groups:

- Information retrieval methods: It is 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: These 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: These methods use sampling of assets to retrieve candidates. They have very high precision and high recall.
- Denotational semantic methods: These methods use signature methods for retrieving candidates. The have very high precision and high recall.
- Structural methods: These methods deal with the structure of the components to retrieve candidates. This group has very high precision and recall.
- Topological methods: This is an approach to minimize the distance between query and reusable candidates. It is difficult to estimate or define recall and precision for these methods [33].

Although [33] proposed six groups, the last one relies on a *measurable* retrieval techniques so it can be considered as an approach for ranking the candidate components of a query [21].

In the work [21] several retrieval techniques were compared. They compared signature matching, text-based, name-based and interface-driven. These techniques were tested using Merobase search engine. The result they obtained was that the best technique is interface-driven in terms of precision.

Another technique that was presented in [19], is test-driven search. This technique ruses 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. Below we will explain with further details how this techniques works.

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 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 [19]. It is a technique that emerges as a solution to the problem of retrieving so many 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 user (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.

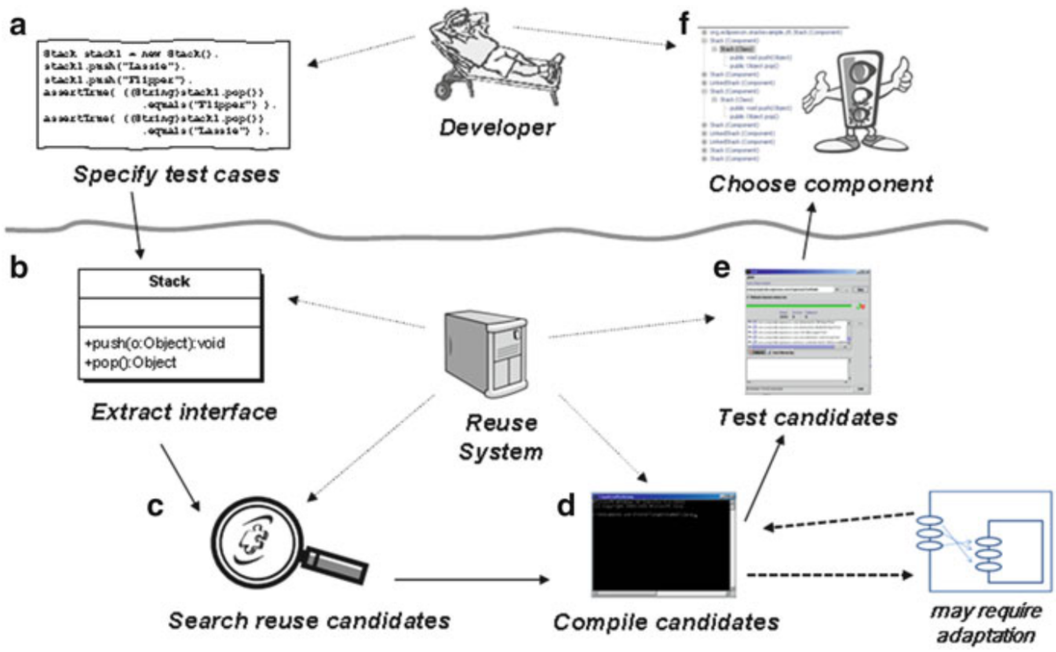

Figure 2.1.: The test-driven reuse *cycle* extracted from [20]

An implementation of this *cycle* is Code Conjurer which can be found in [22] and[20]. 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 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 [40] and CodeGenie [29].

This approach for searching candidates is very promising because of popularity of Agile practices and specifically of Test-Driven development (TDD) [3]. 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.

In spite of the fact that this search method retrieves more relevant resources, it needs improvement in order to be applicable for the daily work of a developer. The amount of time taken from submitting the test case till 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

2.1.5.1. SOCORA

The ranking component approach presented in [25] 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 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 [25]. The figure 2.2 is a schematic illustration of how the component ranking works.

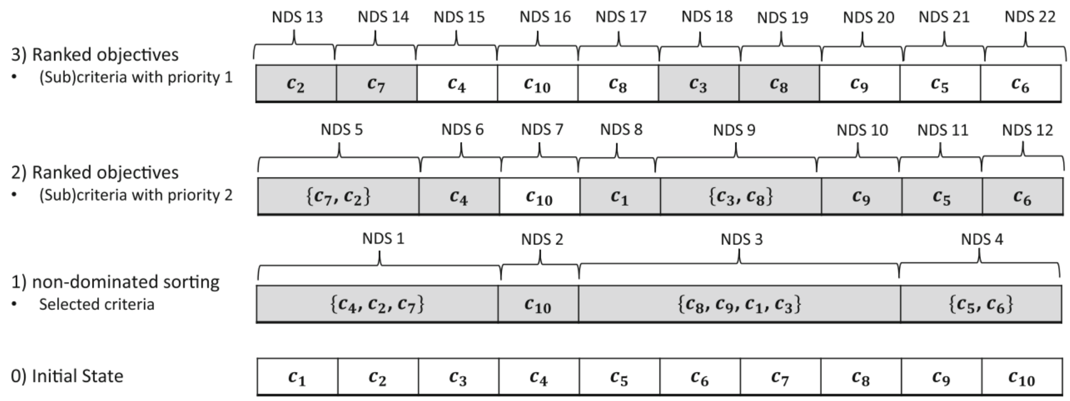

Figure 2.2.: Schematic illustration of SOCORA ranking approach [25]

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[9] 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

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

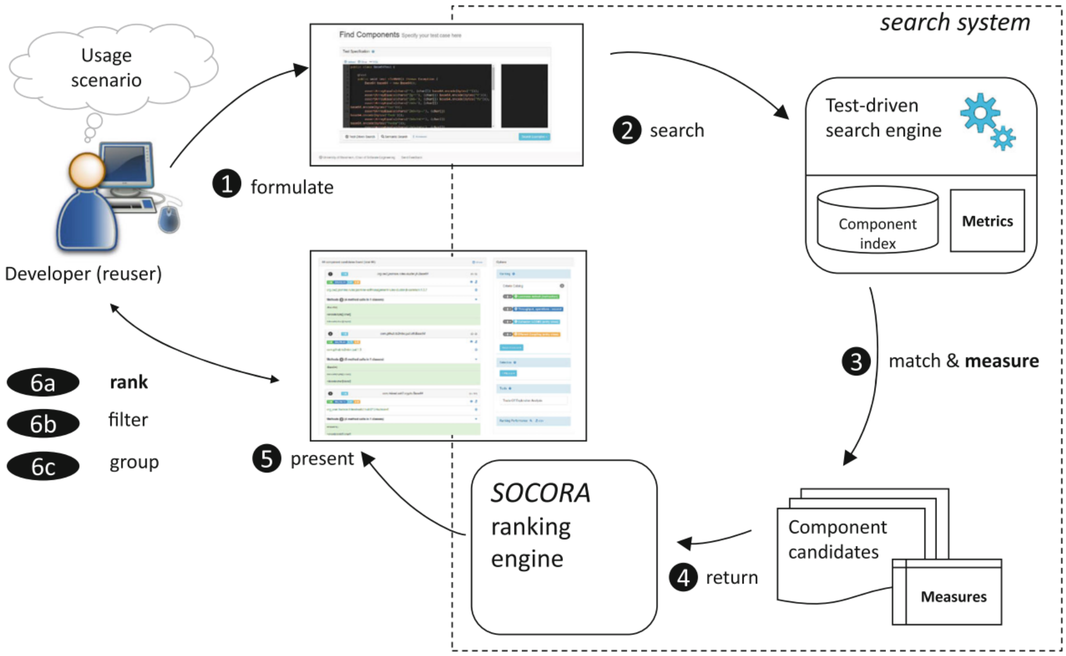

Figure 2.3.: SOCORA and Merobase integration (extracted from [25])

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).

2.2. Continuous Integration

In this chapter we will explain what Continuous Integration (CI) means, we will describe the process steps, its benefits and challenges. Finally we will explain how CI can be leveraged in order to improve Software Reuse.

2.2.1. What is Continuous Integration?

The term can be found in the context of micro-processes development in the work of [4]. Then it was adopted by Kent Beck in his definition of Extreme Programming [2]. However, it was Martin Fowler who was credited with establishing the current definitions of the practices. Fowler defines CI as following:

CI 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[9].

Continuous Integration emerges as a solution for the painful moment of software integration. Although the process of integrating software is not a problem for a one-person project, when it increases in complexity it becomes more problematic. For example, 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 [7].

This step of module integration was a 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.

Therefore we can break up the CI workflow in the following steps [9]:

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 continually 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.

2.2.1.1. 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 VCS are Git, Mercurial and SVN. On top of them, we can find Version Control Platforms (VCP) such as Github, Gitlab or Bitbucket. 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.

Git Within the several different VCSs available, Git is one of the most popular[10]. Git emerges in 2005 as an alternative to BitKeeper[11] after the break

[10] https://rhodecode.com/insights/version-control-systems-2016 (accessed: 14.01.2017)
[11] https://www.bitkeeper.com/ (accessed: 13.01.2017)

of the relationship between the commercial company behind BitKeeper and the community that developed the Linux kernel [5].

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 [5]. The figure 2.4 depicts these three states.

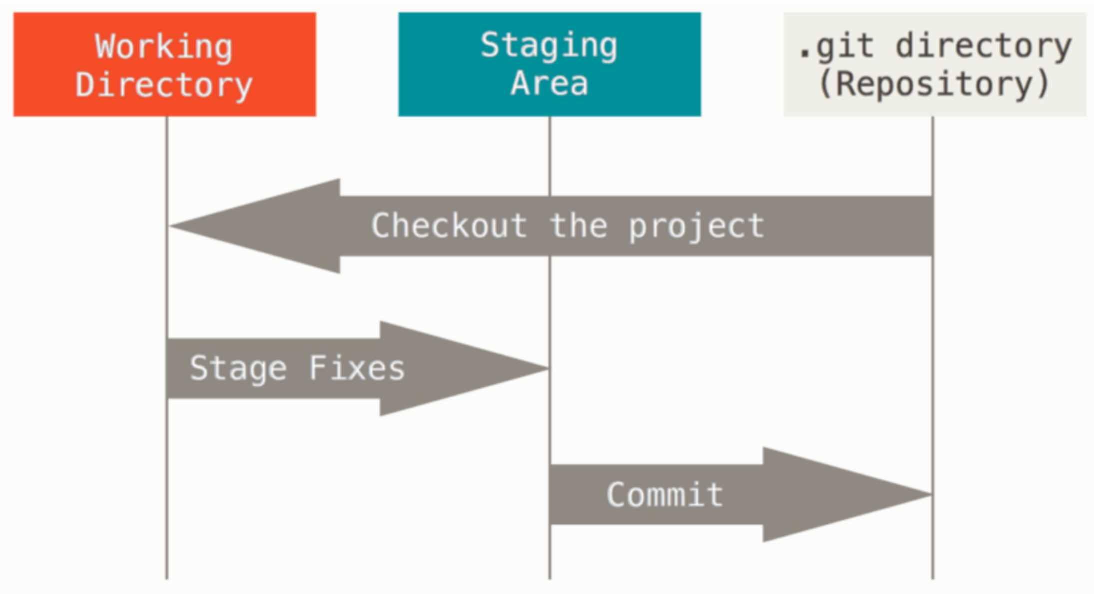

Figure 2.4.: The three main states of Git [5]

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.

Github Github[12] 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 [48] and it was officially lunch on the 10th of April in 2008. Since then, Github has gain popularity through the community reaching more than 50 million projects hosted nowadays.

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[13].

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[14].

2.2.2. 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 [41]. It is important to point out that these benefits come along with other practices such as agile transformation [28] and lean software development [38].

- 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 [36]. 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 [30]. Furthermore, with frequent releases is possible to experiment with new features easily and with low impact [36].

[12] https://github.com/ (accessed: 13.01.2017)
[13] http://archive.is/fl45 (accessed: 13.01.2017)
[14] https://jenkins.io/press/ (accessed: 13.01.2017)

- 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 [31] and the link between software quality improvement and the heavy reliance on automated test combined with smaller more manageable releases [30].
- Improved release reliability: In the work of [36] 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 [11]. In fact, reduction in the stress of developers and other stakeholder have been found by adopting CI [36] [6].
- Improved developer productivity: By automating deployment process, environment configuration and other non-value adding tasks, significant time saving for developers have been observed [42]. Also, in the work of [18] 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.

2.2.3. 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 [42]. 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 [42].
- 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 [42].
- 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 [30].
- 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 [30].
- 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 [30].

2.2.4. Successful CI practices

In order to reduce the risks in adopting CI practices, eight principles, based on several literature reviews, were defined in [41] 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 [[42]; [31]; [44]]. They can be viewed almost as a design pattern for adopting CI/CD, offering a generic solution adaptable to most cases.

2.3. Microservices

Microservices is a software architecture is usually linked to Martin Fowler in its article [12]. 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 [12].

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 **??**): 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.

2.3.1. Characteristics of a Microservice Architecture

In the work of Fowler and Lewis [12] described 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 [12].

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 [10].

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 [12], the core of The Guardian website [15] was built as a monolith, however every new feature has been added as a microservice that uses the core's API.

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

3. Usage Scenario

Before explaining the approach we will propose in the next chapter, we will explain a scenario that illustrates the opportunity found that motivated our approach to improve software reuse leveraging continuous integration. The following scenario is based on the usage scenario found in [25].

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 [15]). Also this team has adopted continuous integration, thus every member of the team is committing code at least once a day and the code committed is tested before it is accepted by the VCS. We will assume that the team is using Git as VCS, Jenkins as CI server, Java as main language and that the team has adopted TDD.

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 team knows that at the moment they are developing. Therefore it would be very useful that an automatic tool analyzes the code and extracts the method signature so it would be able to make a code search for similar methods.

How might we seize continuous integration in order to improve software reuse?
Continuous integration seems to be a very good opportunity for software reuse. According to CI workflow explained in 2.2.1 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 could also extract the test classes to search for similar components.

4. Simile

In this work we present Simile, which is a tool that leverages the Continuous Integration process in order to recommend similar components that are being developed in a software project. Following the usage scenario explained previously in section 3, we will describe how our approach works:

Once a member of the team makes a commit and pushes to the remote Git server, our tool will be triggered by Jenkins. This tool will receives the repository URL where the project is being stored. Then it will clone the repository locally and will go through the source code in order to extract the different class names, method signatures and test classes of the source code. Once done it will make requests to SOCORA using the information extracted. SOCORA will response with all the components that are similar to the components extracted from the project. Simile will get this response and it will generate a report listing all the similar components to the components that are being developed by the team. This report will be sent by email to the members of team so they will know if there is already a component that can be reuse instead of developing it from scratch. Figure **??** describes the process of our approach.

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

4.1. 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.

4.1.1. Simile

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

```
@Async
public void searchForComponents(String repo, String branch, String folder) {
  Cloner cloner = new Cloner(repo, branch, folder);

  cloner.cloneRepository();

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

  File testDir = new File(String.format("%s/src/test", folder));
  JavaClassHandler jch2 = new JavaClassHandler();
  new DirectoryExplorer(jch2, new JavaClassFilter()).explore(testDir);

  socoraRequester.searchComponent("SocoraRequester(searchComponent(String,String):void;)", SocoraRequester.INTERFACE_DRIVEN_SEARCH
```

```
        );
16 }
```

Listing 4.1: Method searchForComponents of Simile.java

First, we clone the project in the server (lines 3 - 5). Then we analyses the Java main code of the project cloned (lines 7 - 9) and the test classes (lines 11 - 13). When we analyse the code we extract interface signatures and test classes, we will explain how we do this in section 4.1.3. With the result of the previous step, we make the requests to SOCORA in order to get the 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.

4.1.2. Cloner

The Java class Cloner.java [A.2] is in charged of cloning the project in a local directory. The listing 4.2 represents the methods cloneRepository.

```
public int cloneRepository() throws IOException {
  int exitVal = 0;
  try {
    String command = this.setCommand(repo, branch, folder);
    Process p = Runtime.getRuntime().exec(command);
    StreamGobbler errorGobbler = new StreamGobbler(p.getErrorStream
        (), "INFO");
    errorGobbler.start();
    exitVal = p.waitFor();
    return exitVal;
  } catch (InterruptedException e) {
    e.printStackTrace();
  }
  return exitVal;
}
```

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*.

4.1.3. Interface signature and test classes extraction

In this section we will explain how our prototype explores the project cloned , and how it extracts the interface signatures and the test classes from the code.

The Java class JavaClassHandler.java [A.3] is in charge of parse the Java classes of the project to which we will search similar components for. This class is an implementation of the interface *FileHandler* [A.4]. The method implemented is *handle* (listing 4.3).

```java
@Override
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.
        getMQLNotation(jcv.getClassObj().getNameAsString(), method.
        getNameAsString(), method.getParameters(), method.getType().
        toString())));
    testClasses.addAll(jcv.getTestClasses());
  } catch (IOException e) {
    new RuntimeException(e);
  }
}
```

Listing 4.3: Method handle of JavaClassHandler.java

This method accepts three parameters: *level*, *path*, and *file*. *level* represents the level of the directory. *path* represents the path of the current directory or

4.1.4. HTTP Controller

The Java class EntryController.java [A.5] 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 4.4 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.

```
@RequestMapping(name = "/repository", method = RequestMethod.POST)
public ResponseEntity<?> setupRepository(
  @RequestParam(name = "repo") String repo,
  @RequestParam(name = "branch") String branch,
  @RequestParam(name = "email") String email) {

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

Listing 4.4: 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 [1] 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.

4.1.5. Cloner

4.1.6. Mailer

4.2. 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

[1] CUID: Collision-resistant ids - `https://github.com/graphcool/cuid-java` (accessed: 04.02.2017).

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 [2] 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 **??**). 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.

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.

4.3. SOCORA Integration

Our prototype uses SOCORA to get similar components. After extracting the interface signatures and test classes from a 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 to Simile.

Simile communicates with SOCORA by making HTTP requests.

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

5. Discussion, Conclusions, and Further Work

The purpose of this book is to understand the influence of representations on the performance of genetic and evolutionary algorithms. This chapter summarizes the work contained in this study and lists its major contributions.

5.1. Discussion

This is the final section **??**. We started in Chap. **??** by providing the necessary background for examining representations for GEAs. Researchers recognized early that representations have a large influence on the performance of GEAs. Consequently, after a brief introduction into representations and GEAs, we discussed how the influence of representations on problem difficulty can be measured. The chapter ended with prior guidelines for choosing high-quality representations. Most of them are mainly based on empirical observations and intuition and not on theoretical analysis.

Therefore, we presented in Chap. **??** three aspects of a theory of representations for GEAs. We investigated how the locality, scaling, and locality of an encoding influences GEA performance. The performance of GEAs is determined by the solution quality at the end of a run and the number of generations until the population is converged. Consequently, for redundant and exponentially scaled encodings, we presented population sizing models and described how the time to convergence is changed. Furthermore, we were able to demonstrate that high-locality encodings do not change the difficulty of a problem; in contrast, when using low-locality encodings, on average, the difficulty of problems changes. Therefore, easy problems become more difficult and difficult problems become easier by the use of low-locality encodings. For all three properties of encodings, the theoretical models were verified with empirical results.

5.2. Conclusions

We summarize the most important contributions of this work.

Framework for design and analysis of representations (and operators) for GEAs. The main purpose of this study was to present a framework which describes how genetic representations influence the performance of GEAs. The performance of GEAs is measured by the solution quality at the end of the run and the number of generations until the population is converged. The proposed framework allows us to analyze the influence of existing representations on GEA performance and to develop efficient new representations in a theory-guided way. Furthermore, we illustrated that the framework can also be used for the design and analysis of search operators, which are relevant for direct encodings. Based on the framework, the development of high-quality representations remains not only a matter of intuition and random search but becomes an engineering design task. Even though more work is needed, we believe that the results presented are sufficiently compelling to recommend increased use of the framework.

Redundancy, Scaling, and Locality. These are the three elements of the proposed framework of representations. We demonstrated that these three properties of representations influence GEA performance and presented theoretical models to predict how solution quality and time to convergence changes. By examining the redundancy, scaling, and locality of an encoding, we are able to predict the influence of representations on GEA performance.

The theoretical analysis shows that the redundancy of an encoding influences the supply of building blocks (BB) in the initial population. r denotes the number of genotypic BBs that represent the best phenotypic BB, and k_r denotes the order of redundancy. For synonymously redundant encodings, where all genotypes that represent the same phenotype are similar to each other, the probability of GEA failure goes either with $O(\exp(-r/2^{k_r}))$ (uniformly scaled representations) or with $O(\exp(-\sqrt{r/2^{k_r}}))$ (exponentially scaled representations). Therefore, GEA performance increases if the representation overrepresents high-quality BBs. If a representation is uniformly redundant, that means each phenotype is represented by the same number of genotypes, GEA performance remains unchanged in comparison to non-redundant encodings.

The analysis of the scaling of an encoding reveals that non-uniformly scaled representations modify the dynamics of genetic search. If exponentially scaled representations are used, the alleles are solved serially which increases the overall time until convergence and results in problems with genetic drift but allows rough approximations of the expected optimal solution after a few generations.

We know from previous work that the high locality of an encoding is a necessary condition for efficient mutation-based search. An encoding has high locality if neighboring phenotypes correspond to neighboring genotypes. Investigating the influence of locality shows that high-locality encodings do not change the difficulty of a problem. In contrast, low-locality encodings, where phenotypic neighbors do not correspond to genotypic neighbors, change problem difficulty and make, on average, easy problems more difficult and deceptive problems easier. Therefore, to assure that an easy problem remains easy, high-locality representations are necessary.

5.3. Further Work

What are the open questions? What should be done next?

Bibliography

[1] Veronika Bauer and Antonio Vetro'. Comparing reuse practices in two large software-producing companies. *Journal of Systems and Software*, 117:545–582, 2016.

[2] Kent Beck. *Extreme Programming Explained: Embrace Change*. Number c. 1999.

[3] Kent Beck. Test-Driven Development By Example. *Rivers*, 2(c):176, 2003.

[4] 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.

[5] Scott Chacon. Pro Git. *Control*, pages 1–210, 2009.

[6] Lianping Chen. Towards Architecting for Continuous Delivery. In *2015 12th Working IEEE/IFIP Conference on Software Architecture*, pages 131–134. IEEE, may 2015.

[7] Paul Duvall, Steve Matyas, and Andrew Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. 2007.

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

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

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

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

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

[13] W.B. Frakes and Kyo Kyo Kang. Software reuse research: status and future. *IEEE Transactions on Software Engineering*, 31(7):529–536, jul 2005.

[14] William Frakes and Carol Terry. Software reuse: metrics and models. *ACM Computing Surveys*, 28(2):415–435, jun 1996.

[15] 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.

[16] 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.

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

[18] Jezz Humble and David Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. 2010.

[19] 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.

[20] 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.

[21] 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.

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

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

[24] Katsuro Inoue, Reishi Yokomori, Tetsuo Yamamoto, Makoto Matsushita, and Shinji Kusumoto. Ranking significance of software components based on use relations. *IEEE Transactions on Software Engineering*, 31(3):213–225, 2005.

[25] Marcus Kessel and Colin Atkinson. Ranking software components for reuse based on non-functional properties. *Information Systems Frontiers*, pages 1–29, jul 2016.

[26] 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.

[27] Charles W. Krueger and Charles W. Software reuse. *ACM Computing Surveys*, 24(2):131–183, jun 1992.

[28] Maarit Laanti, Outi Salo, and Pekka Abrahamsson. Agile methods rapidly replacing traditional methods at Nokia: A survey of opinions on agile transformation. *Information and Software Technology*, 53(3):276–290, 2011.

[29] Otávio Augusto Lazzarini Lemos, Sushil Krishna Bajracharya, and Joel Ossher. CodeGenie: a Tool for Test-Driven Source Code Search. *OOPSLA 2007*, 07pp:525–526, 2007.

[30] 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. *IEEE Software*, 32(2):64–72, mar 2015.

[31] 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. *Empirical Software Engineering*, 20(5):1384–1425, oct 2015.

[32] M.D. McIlroy. Mass Produced Software Components. *Software Engineering: Report on Conference Sponsored by NATO*, pages 138–155, 1968.

[33] a. Mili, R. Mili, and R.T. Mittermeir. A survey of software reuse libraries. *Annals of Software Engineering*, 5(1):349–414–414, 1998.

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

[35] M. Morisio, M. Ezran, and C. Tully. Success and failure factors in software reuse. *IEEE Transactions on Software Engineering*, 28(4):340–357, apr 2002.

[36] 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.

[37] Thomas P. Pole and W.B. Frakes. An Empirical Study of Representation Methods for Reusable Software Components. *IEEE Transactions on Software Engineering*, 20(8):617–630, 1994.

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

[39] Rubén Prieto-Díaz and Peter Freeman. Classifying Software for Reusability. *IEEE Software*, 4(1):6–16, 1987.

[40] Steven P. Reiss. Semantics-based code search. In *Proceedings - International Conference on Software Engineering*, pages 243–253, 2009.

[41] Kim Rejström. Implementing Continuous Integration in a Small Company: A Case Study. 2016.

[42] 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. *Journal of Systems and Software*, 123:263–291, 2016.

[43] RC Seacord. Software engineering component repositories. *Proceedings of the International Workshop on*, 1999.

[44] 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.

[45] Thomas A. Standish. An Essay on Software Reuse. *IEEE Transactions on Software Engineering*, SE-10(5):494–497, 1984.

[46] Kathryn T. Stolee, Sebastian Elbaum, and Matthew B. Dwyer. Code search with input/output queries: Generalizing, ranking, and assessment. *Journal of Systems and Software*, 116:35–48, 2016.

[47] Suresh Thummalapenta and Tao Xie. Parseweb: a programmer assistant for reusing open source code on the web. *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pages 204–213, 2007.

[48] 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.

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

import com.google.common.base.Strings;
import com.sparkpost.exception.SparkPostException;
import de.unimannheim.informatik.swt.simile.services.*;
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;

@Service
public class Simile {

        @Autowired
        private SocoraRequester socoraRequester;

        @Async
        public void searchForComponents(String repo, String branch,
            String folder) throws IOException, InterruptedException,
            SparkPostException {
                Cloner cloner = new Cloner(repo, branch, folder);

                cloner.cloneRepository();

                File maintDir = new File(String.format("%s/src/main"
                    , folder));
                JavaClassHandler jch = new JavaClassHandler();
                new DirectoryExplorer(jch, new JavaClassFilter()).
                    explore(maintDir);
```

```
                    File testDir = new File(String.format("%s/src/test",
                        folder));
                    JavaClassHandler jch2 = new JavaClassHandler();
                    new DirectoryExplorer(jch2, new JavaClassFilter()).
                        explore(testDir);

                    System.out.println(String.format("Methods found in 
                        project: %s", jch.getMethods().size()));
                    System.out.println(String.format("Test classes found
                         in project: %s", jch2.getTestClasses().size()));

                    System.out.println(String.format("Test class to 
                        search"));
                    System.out.println(Strings.repeat("=", "Test class 
                        to search".length()));
                    System.out.println(jch.getMethods().get(0));
                    socoraRequester.searchComponent("SocoraRequester(
                        searchComponent(String,String):void;)",
                        SocoraRequester.INTERFACE_DRIVEN_SEARCH);
          }

}
```

Listing A.1: Simile.java

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

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

import java.io.IOException;

@RequiredArgsConstructor
public class Cloner {

          @Getter
          private final String repo;
          @Getter
          private final String branch;
          @Getter
          private final String folder;
```

```
    public int cloneRepository() throws IOException {
        int exitVal = 0;
        try {
            String command = this.setCommand(repo,
                branch, folder);
            Process p = Runtime.getRuntime().exec(
                command);
            StreamGobbler errorGobbler = new
                StreamGobbler(p.getErrorStream(), "INFO")
                ;
            errorGobbler.start();
            exitVal = p.waitFor();
            return exitVal;
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        return exitVal;
    }

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

}
```

Listing A.2: Cloner.java

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

import com.github.javaparser.JavaParser;
import com.github.javaparser.ast.NodeList;
import com.github.javaparser.ast.body.Parameter;
import com.google.common.base.Strings;
import lombok.Getter;
import org.apache.commons.lang3.StringUtils;
```

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

public class JavaClassHandler implements FileHandler {

        @Getter
        private List<String> methods = new ArrayList<>();
        @Getter
        private List<String> testClasses = new ArrayList<>();

        @Override
        public void handle(int level, String path, File file) {
                System.out.println(path);
                System.out.println(Strings.repeat("=", path.length()));
                try {
                        JavaClassVisitor jcv = new JavaClassVisitor();
                        jcv.visit(JavaParser.parse(file), null);
                        jcv.getMethods().forEach(method -> methods.add(this.getMQLNotation(jcv.getClassObj().getNameAsString(), method.getNameAsString(), method.getParameters(), method.getType().toString())));
                        testClasses.addAll(jcv.getTestClasses());
                        System.out.println();
                } catch (IOException e) {
                        new RuntimeException(e);
                }
        }

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

```java
                return String.format("%s(%s(%s):%s;)", classname,
                    methodName, StringUtils.join(paramTypes, ','),
                    returnType);
        }
}
```

Listing A.3: JavaClassHandler.java

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

import java.io.File;

public interface FileHandler {
        void handle(int level, String path, File file);
}
```

Listing A.4: FileHandler.java

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

import com.sparkpost.exception.SparkPostException;
import cool.graph.cuid.Cuid;
import de.unimannheim.informatik.swt.simile.Simile;
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;
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;

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

@RestController
public class EntryController {

        private static final Logger LOG = LoggerFactory.getLogger(
            EntryController.class);
```

```
@Autowired
private Simile simile;

@RequestMapping(name = "/repository", method = RequestMethod
    .POST)
public ResponseEntity<?> setupRepository(@RequestParam(name
    = "repo") String repo,
```

```
            if(validateRequest(repo, email).isPresent()) {
                LOG.debug("The request is not valid. Request
                    -> repo: %s - branch: %s - email: %s",
                    repo, branch, email);
                return ResponseEntity.badRequest().body(
                    validateRequest(repo, email).get());
            } else {
                LOG.debug("The request is valid. Request ->
                    repo: %s - branch: %s - email: %s", repo,
                     branch, email);
                simile.searchForComponents(repo, branch,
                    Cuid.createCuid());
                return ResponseEntity.ok(new Message("
                    Repository set successfully", 200));
            }
        }
```

```java
private Optional<Message> validateRequest(String repo,
    String email) {
        LOG.debug("Validating required parameters in request
            ");
        StringBuilder str = new StringBuilder();
        if(repo.isEmpty()) str.append("repo is required");
        if(email.isEmpty()) str.append("email is required");
        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()) {
                return Optional.ofNullable(new Message(str.
                    toString(), 400));
        }
        return Optional.empty();
}

/**
 * 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.
 * */
private boolean isValidGithubWebURL(String gitRepo) {
        final Pattern pattern = Pattern.compile("(http(s)?)
            (:(//)?)([\\w\\.@\\:/\\-~]+)(\\.git)?");
        return pattern.matcher(gitRepo).matches();
}

/**
 * Validates if an email address is valid.
 *
 * @see <a href="http://www.regexr.com/3c0ol">http://www.
 *    regexr.com/3c0ol</a>
 *
 * @param email email address to be validated.
 * @return true in case the email address is valid,
 *    otherwise false.
 * */
private boolean isValidEmail(String email) {
```

```java
            final Pattern pattern = Pattern.compile("[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:.[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*@(?:[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.5: EntryController.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.

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 [23].

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.

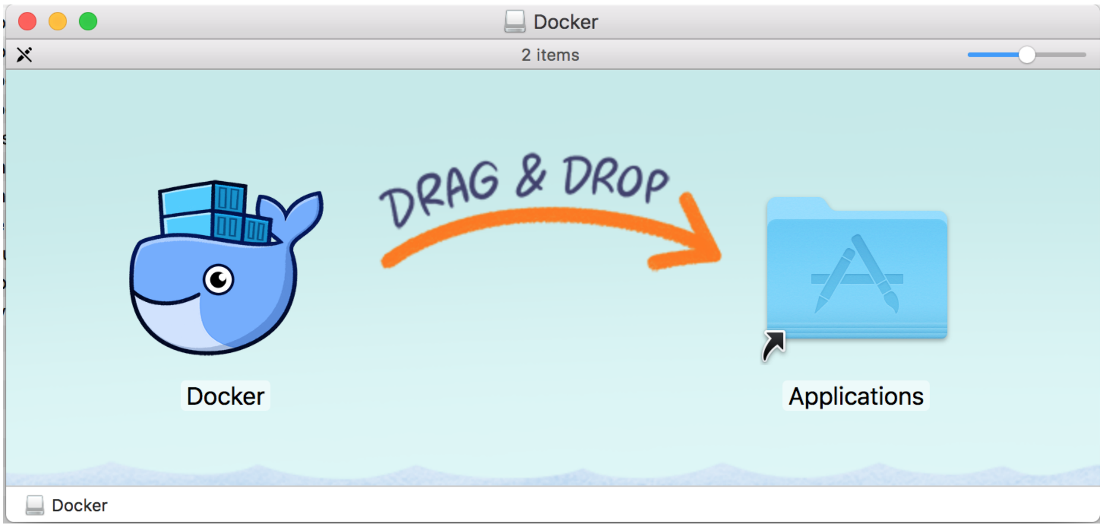

Figure B.1.: Docker for Mac installation

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.

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

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

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

Then we add the official Docker's GPG key.

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

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

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

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

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

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

Then we install docker.

```
$ 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.

```
$ 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.

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

```
latest: Pulling from library/jenkins

# some irrelevant output was remove
Digest: sha256:5046434030be395ec977c98e11...
Status: Downloaded newer image for jenkins:latest
```

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

```
$ docker run -p 8080:8080 -p 50000:50000 jenkins
  # some irrelevant output was remove
  *************************************************************

  Jenkins initial setup is required. An admin user has been
  created and a password generated.
  Please use the following password to proceed to installation:

  95199411f1894bfa97e937147c41aa62 # IMPORTANT, default admin pass

  This may also be found at: /var/jenkins_home/secrets/initial.

  *************************************************************
  # some irrelevant output was removed
  Jan 19, 2017 8:28:05 AM hudson.model.AsyncPeriodicWork$1 run
  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.

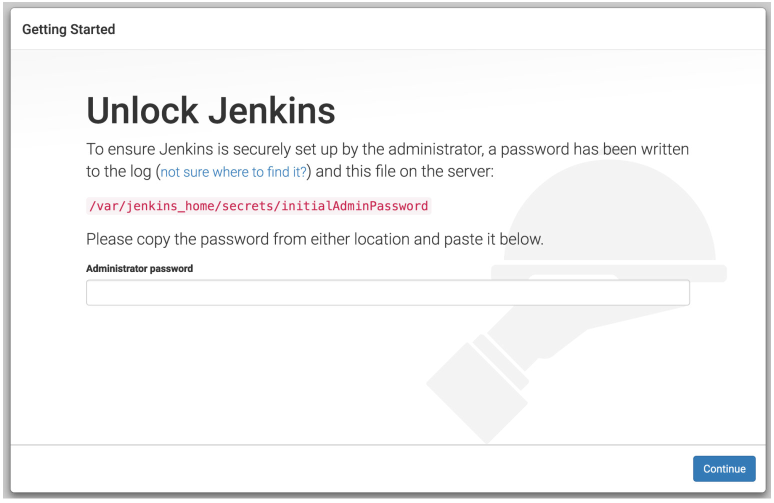

Figure B.2.: First page of Jenkins setup

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.

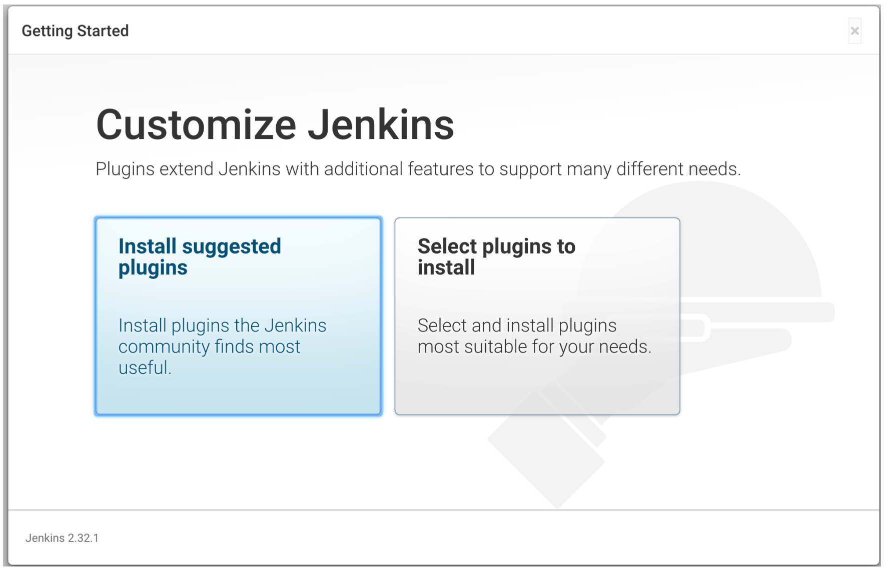

Figure B.3.: Second page of Jenkins setup

Then you should see something like the figure B.4. Here we just need to wait untill the plugins installation finish.

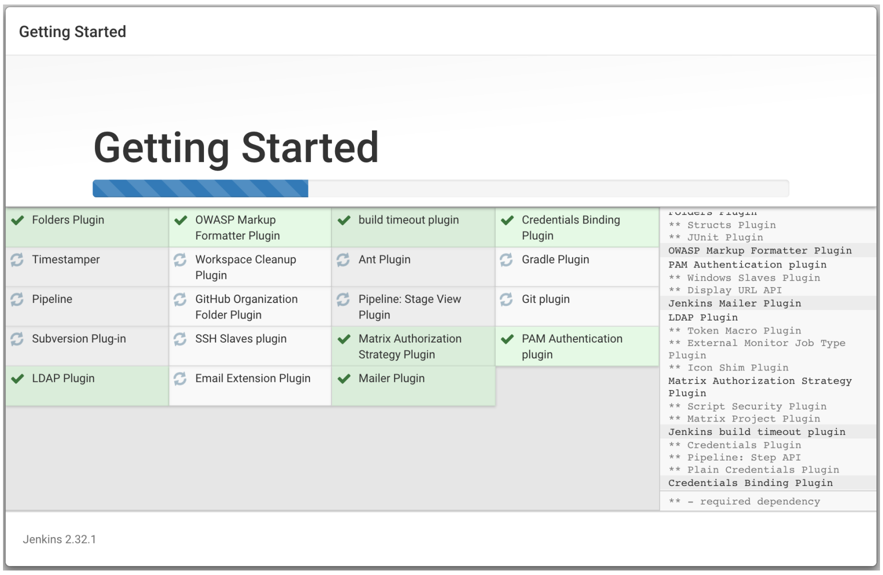

Figure B.4.: Third page of Jenkins setup

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)

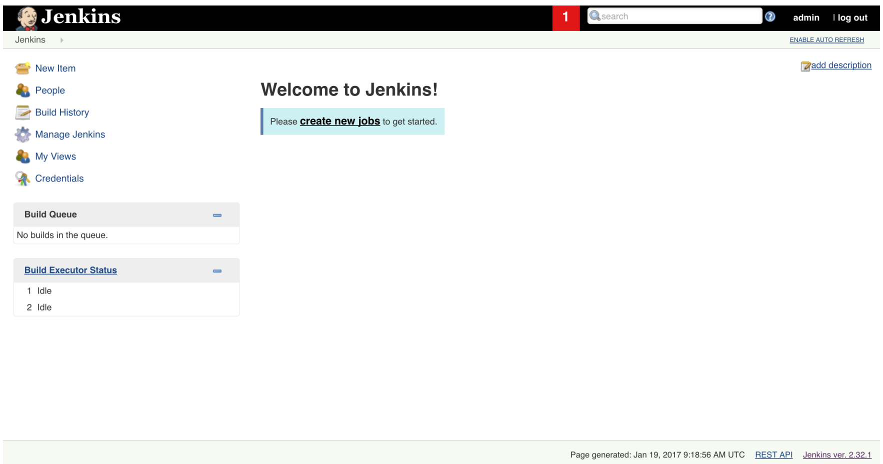

Figure B.5.: Jenkins home page

B.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.6).

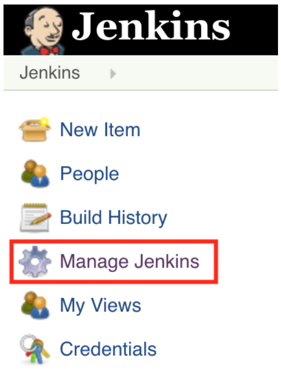

Figure B.6.: Manage Jenkins option

Then click on *Manage Plugins* (figure B.7).

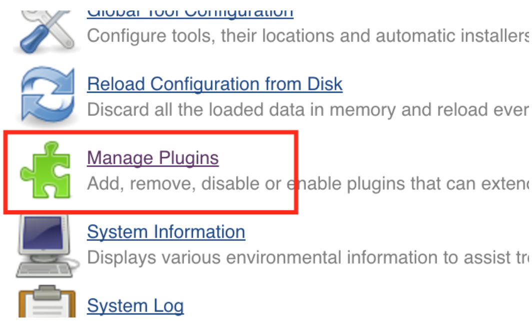

Figure B.7.: Manage Plugins option

Then go to *Advanced*, scroll down until *Upload Plugin* section (figure B.8).

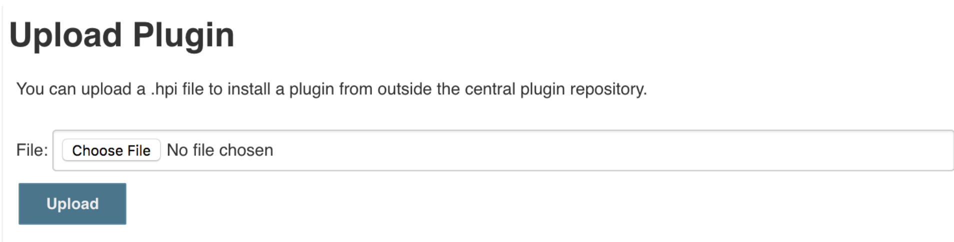

Figure B.8.: Upload Plugin section

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

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

Martin Mustermann