Institute of Software Technology

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Process Analysis No. 9

Analysis of the Software Development Process of inspectIT and Changes Required for Joining the Eclipse Foundation

Thomas Düllmann, Tobias Rudolph, Anton Scherer

Course of Study: Softwaretechnik

Examiner: Prof. Dr. Lars Grunske

Supervisors: Dr.-Ing. André van Hoorn,
Ivan Senic (NovaTec GmbH)

Commenced: 2014/11/03

Completed: 2015/05/05

CR-Classification: D.2.7, D.2.9

Abstract

In todays world there are many arguments for companies to use open source software. On the one hand they profit in terms of guaranteed future from the possibility that the software is supported by the open source community after the developing company decides to stop the support for its product. On the other hand companies can economically profit from the fact that they do not have to pay for using the software.
A good practice to get in touch with these customers that use or plan to use open source software could be developing open source software oneself and publishing these software under a well-known open source license.
One leading player in open source is the *Eclipse Foundation*. Since their establishment in 2004, more and more software products join the *Eclipse Foundation* , often using its open source license Eclipse Public License (EPL). A product that could benefit from becoming an Eclipse project is NovaTec's Application Performance Management (APM) tool *inspectIT*. In this process analysis, we give a summary about the steps a project team needs to make to join the *Eclipse Foundation* and publish software under the EPL.

Contents

Chapter 1

Introduction

1.1 Motivation

A rising number of companies tends to change the way of developing and publishing their software to profit from the growing open source software market [Martin Hörst, 2011]. One way for a company to participate in that market is publishing its own software under a well-known open source software license. The NovaTec Consulting GmbH intends to take the step from a free license model with closed source to an open source license for its APM tool *inspectIT*. The option which is analyzed in this study to accomplish that goal is to publish *inspectIT* as an Eclipse project. One main point, why NovaTec Consulting GmbH considers the option to publish its APM tool as an Eclipse project is that with this step, *inspectIT* could get a higher popularity compared to the status quo. Currently there is no APM tool which is comparable with *inspectIT* in the Eclipse cosmos. So they would be a trend setter and could profit from that role. With that higher population NovaTec Consulting GmbH could offer consulting for their APM tool, give courses and offer to develop enhancements for *inspectIT*. Another argument for joining Eclipse is that *inspectIT* is already based on the Eclipse Integrated Development Environment (IDE). Becoming an Eclipse project would be a consequent next step. Last but not least, customers have the possibility to contribute to the project. On the customers' side, they can assure that there is the possibility to support the tool on their own, even if NovaTec Consulting GmbH would decide to stop supporting *inspectIT*. With that advantage, more customers could consider using *inspectIT*. On the developers' side, there is the possibility that some customers start contributing to *inspectIT*. So NovaTec Consulting GmbH could get help for the development of *inspectIT* for free. In order to do so, NovaTec Consulting GmbH has to analyze its current software development process and compare it with the requirements defined by the *Eclipse Foundation* . During this assessment, possible problems in the general process should be detected and changes that are required to comply with the *Eclipse Foundation* requirements should be identified.

1.2 Goals

Originally, NovaTec Consulting GmbH wants to use the results of this process analysis to see if it is possible to migrate *inspectIT* to the *Eclipse Foundation* . So the NovaTec Consulting

GmbH expected knowledge about what parts of the development process has to change to be able to join the *Eclipse Foundation* .
During our process analysis NovaTec Consulting GmbH decided not to join the *Eclipse Foundation* .

As a consequence, the focus of the analysis shifted towards a more general inspection on how a project could become part of the *Eclipse Foundation* . Furthermore, advantages and disadvantages of a software which joins the *Eclipse Foundation* should be mentioned in this documentation. As a second result we will present a short case study to show what NovaTec Consulting GmbH would have had to do if they decided to become part of the *Eclipse Foundation* in the future. With that knowledge they can do that step easier if it might become relevant to them at some point of time.

Another goal of this study is to analyze and document the software development process and environment of *inspectIT* such as the development-, build-, and release process. During this analysis some suggestions of improvements can be made, although that is not the main focus of our work.

1.3 Document Structure

The remainder of this document is structured as follows:

- ▷ In Chapter 2 we give an overview about NovaTec GmbH and a quick introduction into a process analysis in general. Furthermore we will discuss our project plan in that section.
- ▷ Chapter 3 gives an overview of the requirements regarding the joining process and the processes within the *Eclipse Foundation* .
- ▷ In Chapter 4 we document the software engineering process of *inspectIT* and compare two projects which planed to become part of the *Eclipse Foundation* .
- ▷ Finally, in Chapter 5, we summarize our process analysis and give a personal feedback about the EPL and the consequences becoming part of the *Eclipse Foundation* as well.

Chapter 2

Research procedure

This chapter contains a short description of the NovaTec GmbH. Furthermore an overview of the process analysis is given.

2.1 Company Background

NovaTec GmbH is an IT consulting company with its headquarters based in Leinfelden-Echterdingen, Germany. Besides the headquarter, NovaTec is also located in Frankfurt, Berlin, Munich and Jeddah (Saudi Arabia). The company was founded in 1996 and has grown since. Different services are provided by NovaTec. This includes consulting with focus on application performance management, agile methods, agile quality engineering, data center automation, enterprise application integration, business process management and enterprise application development. Since its foundation, NovaTec GmbH accomplished projects with many partners in different sectors like financial services, health care, logistics, defense and security, automotive and industry and public sector. NovaTec GmbH is divided into three independent operation units: The NovaTec Solution GmbH, the NovaTec Products GmbH and the NovaTec Consulting GmbH.

As an independent consulting company, NovaTec Consulting GmbH is a licensed partner of many popular IT companies. Besides these partnerships, NovaTec is also developing some own IT tools. One example of a software developed by NovaTec Consulting GmbH is the APM tool *inspectIT*. At this time, the APM tool is closed source but freely available. The business model of *inspectIT* is mainly focused on consulting services. So NovaTec Consulting GmbH would not lose any license fees if *inspectIT* would become an Eclipse project. The main benefit for NovaTec Consulting GmbH for publishing their tool as an Eclipse project would be a larger distribution *inspectIT* as the APM tool can profit form the popularity of the *Eclipse Foundation* .

2.2 Process Analysis

Based on [Behr and Tyll, 2003], the goal of a process analysis is to analyze a current process and look for improvements to perform this process in a more efficient or better way. An overview is provided in Figure 2.1.

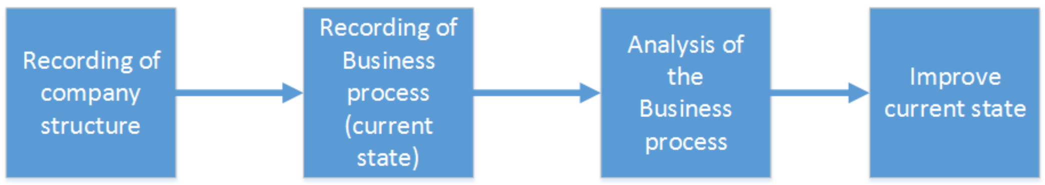

Recording of company structure
Recording of Business process (current state)
Analysis of the Business process
Improve current state

Figure 2.1. representation of the concept of process analysis following.

To achieve this goal, first of all, the status quo has to be analyzed and documented if not done before. Based on these information improvements are suggested to get an better process. There are three different methods described in [Behr and Tyll, 2003] how to gather information about the status quo during a process analysis:

- Survey: If a process analysis is using the survey technique to get information, questions which could show weaknesses of a process have to be collected. As a next step these questions should be asked persons who are involved in the process which is being analyzed. That could be done either in an oral or written form. At the end the results of the survey have to be interpreted.
- Observation: For this method the processes which are analyzed are observed either in a passive or an active way. During a passive observation the observer does not interact with the process, he just observes the process. While an active observer is part of the process. After a specific period of time the experiences which are collected with this method are analyzed.
- Analysis of existent data: If a process analysis works with existent data, mainly available documentation is used for the analysis to suggest improvements of processes.

In this project mainly the third method is used. Therefore the documentation of the *inspectIT* development process and the documentation about the *Eclipse Foundation* are analyzed.

2.3 Project Plan

In this section we describe the individual tasks we defined for this process analysis. To give a overview about these tasks, we provide a Gantt chart in Figure 2.2 to show our time plan as well. The project plan was created as a first step in our project. Not everything has worked out as planned for different reasons. In the following we give a short overview of these task and discuss why some of them did not worked our as planned.

2.3.1 Analysis of inspectIT's current Software Development Process

Analyze the current software development and maintenance processes of *inspectIT*. The result is shown in this document. We achieved this by reading documentation and interviewing a developer of NovaTech Consulting GmbH who is involved in the development of *inspectIT*.

2.3.2 Analysis of the Eclipse Foundation

For publishing a project as an Eclipse project there are requirements and rules designed by the *Eclipse Foundation* . We summarize the requirements to give a rough overview. One result of this step is a checklist which can be applied if initially joining the *Eclipse Foundation* with a project. We also recap the steps needed for publishing a new release for a software project within the *Eclipse Foundation* .

2.3.3 Case Study: inspectIT going Open Source

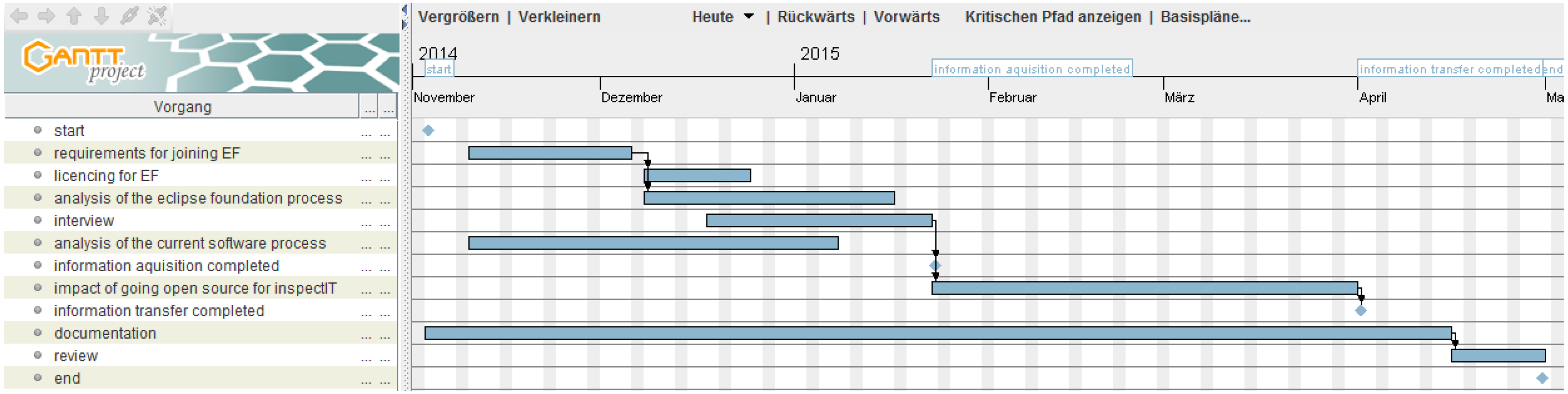
After the collection of the current software process of *inspectIT* and the requirements of joining the *Eclipse Foundation* we document the challenges which would have occured if NovaTech Consulting GmbH had decided to publish *inspectIT* as an Eclipse project. At the point of time we started with that task NovaTec Consulting GmbH decided not to publish *inspectIT* as Eclipse project. As a result of that decision we agreed to give only an overview about what had to be done before publishing *inspectIT* as an Eclipse project.

2.3.4 Interviewing

We looked for potential interview partners. With the help of NovaTec we intended to get in contact with an expert who is the CEO of a company whose software joined the *Eclipse Foundation* in the past. After the decision not to join the *Eclipse Foundation* we lost contact with that person before he could answer our questions about his experience with the *Eclipse Foundation* . But fortunately his company published a case study within some articles in the Eclipse Magazine so we can get information about their experience by reading these articles.

2.3.5 Documentation

The task documentation contains the documentation of the results of the different tasks. This task is done beside the tasks mentioned before. At the end, there is time reserved for the review process.


Vergrößern | Verkleinern
Heute | Rückwärts | Vorwärts
Kritischen Pfad anzeigen | Basispläne...
GANTT project
2014
2015
start
information aquisition completed
information transfer completed
end
November
Dezember
Januar
Februar
März
April
Ma

Vorgang

- start
- requirements for joining EF
- licencing for EF
- analysis of the eclipse foundation process
- interview
- analysis of the current software process
- information aquisition completed
- impact of going open source for inspectIT
- information transfer completed
- documentation
- review
- end

Figure 2.2. Gantt chart

Chapter 3

Fundamentals of the Eclipse Foundation

The *Eclipse Foundation* states in its bylaws [Eclipse Foundation, 2003]: "The Eclipse technology is a vendor-neutral, open development platform supplying frameworks and exemplary, extensible tools (the "Eclipse Platform"). [..] The purpose of Eclipse Foundation Inc., (the "Eclipse Foundation) is to advance the creation, evolution, promotion, and support of the Eclipse Platform and to cultivate both an open source community and an ecosystem of complementary products, capabilities, and services."

This chapter is intended to give a short overview of the requirements regarding the joining process for new open source projects and the expected processes while the project is part of the *Eclipse Foundation* .
The following resources have been used:

- ▻ Committers Bootcamp Presentation Slides (05/2014) [Beaton et al., 2014]
- ▻ Eclipse Development Process 2014 Documentation [Eclipse Foundation, 2014a]
- ▻ Eclipse Wiki - Development Resources [Eclipse Foundation Wiki]
- ▻ Bylaws of Eclipse Foundation [Eclipse Foundation, 2003]
- ▻ Eclipse Intellectual Property Policy [Eclipse Foundation, 2011]
- ▻ Eclipse Public License Agreement [Eclipse Foundation]
- ▻ Eclipse Foundation Contributor License Agreement [Eclipse Foundation, 2013]

3.1 Core Principles

The *Eclipse Foundation* defines three main Rules of Engagement which state the core principles of the *Eclipse Foundation* :

Transparency All information should be accessible without restrictions. This means, that discussions, plans and minutes should be open, public and easily accessible.

Openness Everybody should have the same opportunities to contribute and everybody should participate with the same rules. Nobody is excluded in any way (not even competitors).

Meritocracy The more one participates and contributes the more responsibilities are given to him. Leadership roles are also merit-based (earned by peer acclaim).

3.2 Communities

All people having to do with an *Eclipse Foundation* project can be divided into three groups (non-exclusive):

Users People only using the software from a project.

Adopters Individuals, groups or organizations that use the software from a project to either build their own products or extensions.

Developers Consisting of contributors and committers participating in the project itself.

3.3 Basic Project Structures

To understand the processes and requirements of the *Eclipse Foundation* it is important to have a basic understanding of the structures the *Eclipse Foundation* postulates.

3.3.1 Project Structure and Organization

Eclipse projects are structured in a hierarchical order. Every project, independent from the fact whether it is a *Top Level Project* or a sub-project, has its own resources (especially repositories and issue tracking) and committers. It is possible that projects join existing projects as sub-projects. Some resources can be shared between a parent and a sub-project (e.g, website, downloads, website). Every *Top Level Project* is lead by a Project Management Committee (PMC) who is assigned to keep oversight of the whole underlying project structure and its processes. For each project there is at least one *Project Lead* who is responsible for the specific project and ensures that the *Eclipse Development Process* is implemented correctly and the activities follow the core principles of the *Eclipse Foundation* . In addition to the *Project Lead(s)* there are two groups of people working on a project: Contributors and Committers.

Contributors "Contributors are individuals who contribute code, fixes, tests, documentation, or other work that is part of the project." [Eclipse Foundation, 2014a]

Everybody can contribute to projects after registering an account and signing the Eclipse Contribution License Agreement (CLA) (Eclipse Foundation [2013]). Signing the CLA is required to make sure that all contributors know and follow the Intellectual Property (IP) terms of the *Eclipse Foundation* .

Committers "Committers have write access to the project's resources [..] and are expected to influence the project's development." [Eclipse Foundation, 2014a]

The committers' responsibility is to actively take part in the forthcoming of a project. Committers can nominate Committers-to-be and participate in votings whether they are accepted as a Committer. If they become inactive for at least six months, the Committer state can be revoked.

3.4 Eclipse Public License

The EPL is an open source license, published in 2004. Basically, all current and future members of the *Eclipse Foundation* must conform to this license. Originally, EPL was derived from the Common Public License (CPL). That is why both licenses are very similar. However, EPL is at some points less restrictive regarding patent litigation terms. Although the details of the license model are out of scope, a broad overview of the major aspects will be provided in the following. In general, there are two types of parties defined by the EPL: Persons (or entities) are named *Contributors* if they develop and/or distribute code for the project. This could be either the initial starting code as well as subsequent contributions. Also persons who just redistribute the program are regarded to be contributors. Redistribution means distributing the unmodified or changed software through different channels. In contrast, *Recipients* include anyone receiving the program under the EPL. Contributors must identify themselves as originators of their code, which is why they cannot remain anonymous after code commitment. In general, code may be adapted and distributed under EPL freely without any warranty or conditions. It is even allowed to compile a program licensed under the EPL without modification and commercially license the result in accordance with the terms of the EPL. Moreover, if a person modifies the program but does not distribute it (e.g. only for his/her own purpose), the changes must not be made available to others. Though, if someone does distribute a modified program which originally was under EPL, regardless if it is a free or commercial distribution of the object code, changes must be made open source. But, if something is built on top of the program (existing code is not changed but extended) then it is not required to distribute the added source code. However, there is a fundamental problem when building and distributing software that contains source code under EPL and code under another license. Then, it must be checked carefully if the licenses comply to each other. For example, popular licenses such as GNU General Public License or Berkeley Software Distribution License are not compatible with EPL. In contrast, other licenses such as Apache Software License 2.0, W3C Software License, Common Public License 1.0 are approved for use by third-party code redistributed by Eclipse projects which run under EPL anyway. However, one must be aware that when developing a software product including copies of code licensed under EPL, redistribution is only permitted under the EPL license. Read our own opinion about the EPL in Chapter 6.

3.5 Intellectual Property Policy

The Intellectual Property Policy (IP Policy) defines rules that apply to all content that the *Eclipse Foundation* accepts, redistributes or hosts and to all other aspects regarding intellectual property. This section is based on the Eclipse IP Policy document [Eclipse Foundation, 2011].

3.5.1 Contribution Acceptance (inbound Licensing)

The *Eclipse Foundation* preferably accepts content which is licensed under the EPL. Content (standing for any code/media that is submitted to an either new or existing project) that is not licensed under the EPL has to meet multiple requirements to be able to be accepted:

First, the copyright holder of the content to contribute has to be unwilling to make it available under EPL or the project license is not the EPL. Furthermore, Eclipse Management Organization (EMO), the PMC Leader and the committer have to state that the content in question is important for achieving the project plan. Eventually the Eclipse Foundation Board and the PMC have to review and approve the proposed alternative terms and conditions.

In order to gain the rights to redistribute the submitted content, the *Eclipse Foundation* uses the following mechanisms:

▻ *Committer Agreement*

▻ Explicit license grants

▻ *Eclipse.org Terms of Use*

▻ Other agreements between the submitter and the EMO

The EMO is responsible for the correct processing of submissions in terms of licensing and redistribution and hosting.

3.5.2 Contribution Licensing (outbound Licensing)

In general, Eclipse projects shall be licensed under the EPL. Also material other than software and documentation that is copyrightable (e.g., images) shall be licensed under the terms of the EPL. Exceptions can be made by approval by the Board of Directors (see Eclipse Foundation [2003], Section 3.9).

3.5.3 Due Diligence and Record Keeping

Apart from exceptions, the EMO is held to ensure the compliance with the IP Policy. In doubt of the ability to redistribute content, the committer should contact the PMC and the EMO for assistance with her case. If disputes between PMC and EMO should arise, the PMC has the possibility to appeal to the Eclipse Foundation Board for resolution of the dispute. Both, PMC and EMO are responsible for keeping the information collected by the committers for future reference.

Chapter 4

Consolidation of Requirements of the Eclipse Foundation

4.1 Project Lifecycle

A projects' lifecycle ranges from its creation to its termination. The steps from the beginning to the end are presented in this chapter.

4.1.1 Joining Process

In order to become an Eclipse project, there is a predefined process that has to be followed. This section gives a short overview over these steps.

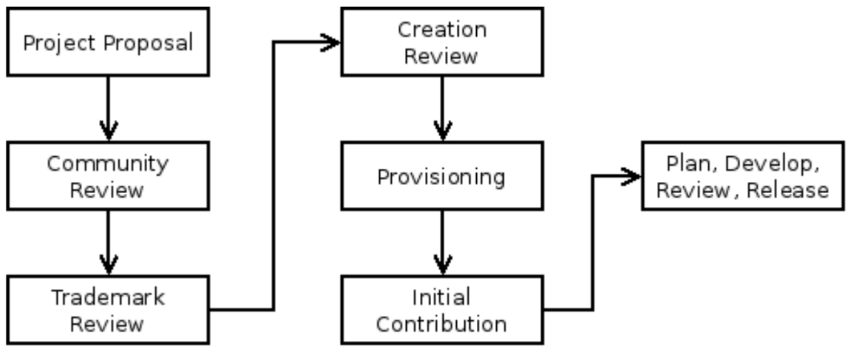


Figure 4.1. The Eclipse project creation process according to Beaton et al. [2014].

4.1.2 Prerequisites

Every project intending to join is expected to have (useful) code already. Eclipse projects are responsible for developing a community of users, adopters and contributors. Having these aspects in mind, people interested in starting an Eclipse project can contact the *Eclipse Foundation* .

Before actively starting to create a new Eclipse project, one should check whether there are similar projects which can be extended in any way to achieve the intended aim. If this is not the case, a project name has to be chosen (according to the HowTo `https://wiki.eclipse.org/Development_Resources/HOWTO/Pre-Proposal_Phase`). Afterwards, the creation of a project proposal can begin.The EMO will assist with the preparation of a project proposal.

Project Proposal

The project proposal is the first public presentation of the project. It should contain information about what the aim of the project is, how it is intended to be reached and what plans currently exist to bring the project on its way. Furthermore, it is often used to attract potential contributors or third parties that might be interested in supporting the project. The proposal is published on the website of the *Eclipse Foundation* . During this phase two mentors from the Architecture Council have to be identified and listed in the project proposal. If no mentors could be identified during this phase, the project lead can file a bug against the Architecture Council to actively request mentors. According to the HowTo (`https://wiki.eclipse.org/Development_Resources/HOWTO/Pre-Proposal_Phase`) some information about the project-to-be are required to be filled in (e.g., background, scope, licenses, committers, mentors). Those mentors assist a project with all questions regarding the creation and the development of the *Eclipse Foundation* processes. Further information about the Architecture Council can be found in the Eclipse Foundation Wiki (`https://wiki.eclipse.org/Architecture_Council`)

Community Review

After the publication of the first version ot the project proposal, the community has at least two weeks to review the proposal. During this phase the creators are held to respond to questions from the community, identify additional committers and refine the proposal.

Trademark Review

The aim of the trademark review is to make sure, that the codebase is reviewed, does not contain inappropriately copied code, that license terms are respected and that third party libraries used in the project are approved by the *Eclipse Foundation* . Eventually, the IP log is submitted (`http://wiki.eclipse.org/Development_Resources/IP_Log`). This process is an offer by the *Eclipse Foundation* to make sure the code base is clean from a jurisdictional point of view. This is a big advantage for a project and would be a very complex and expensive task in case it had to be done by external experts in that field.

Creation Review

The intent of the *Eclipse Foundation* is not to be a "code dump". To prevent this, the creation review makes sure, that the project has sufficient resources to keep it alive and to achieve its aims. Therefore, the project has to nominate its initial committers and justify its choice by stating the relationship of the committers to the project and their history with the project (e.g., previous contributions, technical experience in the field of the project). The aim is to keep experienced contributors in the project and justify the initial committership in the meritocracy-based ecosystem of the *Eclipse Foundation* . Additionally, the creation review phase gives the opportunity to the community for a final project review and pose further questions regarding the project.

Provisioning

The provisioning phase begins with the project lead submitting a provisioning request (`https://www.eclipse.org/projects/project_provisioning_request.php`) to the EMO. After the processing of the provisioning request and after the IP team has handled the committer paperwork, the requested resources are provisioned by the webmaster team. The main resources that are provisioned are:

▻ Version Control System: `git/gerrit`

▻ Issue tracker: `bugzilla`

▻ Forums

▻ Mailing lists

Initial Contribution

The following list shows the steps the initial contribution consists of.
Source: Eclipse Foundation Wiki (`http://wiki.eclipse.org/Development_Resources/Initial_Contribution`)

1. Wait for provisioning notice
2. Ensure namespace aligns with naming conventions (`org.eclipse.<subproject>.<component>[.*]`). See Eclipse Foundation Wiki (`http://wiki.eclipse.org/Naming_Conventions`) for further details.
3. Ensure copyright and license notices have been applied to source. The default copyright and license notice can be found under: `https://www.eclipse.org/legal/copyrightandlicensenotice.php`
4. Ensure availability of the notices (about.html, license files, ..)
5. Pack source code and attach to Bugzilla record as "initial contribution"

6. Use foundation portal to open Contribution Questionnaire (CQ) referencing the record (bug), which leads to the initial review by the *IP Team*.
7. On request add the source code archive file to CQ, too

If there are any third party references, there is an alternative procedure to be followed: http://www.eclipse.org/org/documents/Eclipse_Policy_and_Procedure_for_3rd_Party_Dependencies_Final.pdf

Plan, Develop, Review, Release

The aspects of *Plan, Develop, Review and Release* are covered in Chapter 4.2 in detail.

4.1.3 Project Phases

A project can be in several phases. Those phases are presented in the paragraphs below. This section is based on Eclipse Foundation [2014a].

Incubation If a project is created, it is in the *incubation phase*. This phase aims to give a project time to establish its processes, community and technology to become a functional open source project. While the project is in the *incubation phase* it is allowed to publish releases; the phase ends either with a graduation review (⇒ Mature) or a termination review (⇒ Archived).

Mature The *mature phase* begins, when the project could demonstrate in the graduation review, that it has adopted the *Eclipse Foundation* processes and that it is a functional and working open source project. This phase ends with a termination review (⇒ Archived).

Archived If a project becomes inactive it reaches the *archived phase* after a termination review. This can happen either if the projects aims are reached or the project does not have enough resources (e.g., contributors) to stay active. When in *archived phase*, all project information (e.g, mailing lists, website, repository content) are archived and all active services are disabled (e.g., repository, mailing lists). The project gets a website with a short description, the last release and the archived resources. Finally, the project is removed from the active projects list and added to the archived projects list.

4.1.4 Best Practices

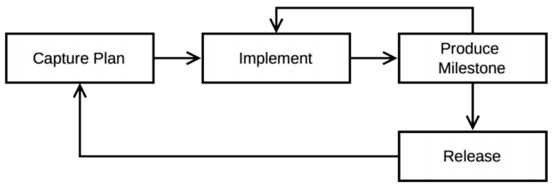
Apart from the adoption of the processes of the *Eclipse Foundation* , the *Eclipse Foundation* recommends to adhere to the following best practices to actively improve and support a project:

- ▻ Careful tracking of all project contributions

- ▷ Anticipate spending a significant amount of time for responding to questions in forums and mailing-lists, stackoverflow.
- ▷ Attend conferences (especially eclipse summit europe, eclipsecon)
- ▷ Presenting webinars
- ▷ Writing papers

4.2 Development Process

The next paragraphs describe the actual process of developing new code after the joining process to *Eclipse Foundation* finished successfully. First of all, each project has one or more project leaders who are responsible that the project's committers follow the defined rules and guidelines. The development team consists of comitters and contributors who together follow the general process illustrated in Figure 4.2.



.

Figure 4.2. General Development Process Beaton et al. [2014].

4.2.1 Capture Plan

The process starts with capturing the project plan for each major and minor project release which should be made public. Doing so encourages the community to get involved into the project. That is why the project plan should be started at the beginning of the release cycle. Ideally, the release is directly linked to the project plan using the "Plan URL" in the project metadata. Project metadata are relatively static structural information such as the project description, the names of the project's mailing lists, historical information such as previous release downloads, IP logs and finally current status and future milestone plans, release dates, etc. In general, the project plan can be specified in two ways: Either as a release record or as an XML-based project plan which can be implemented easily using some XML-Templates where a neat HTML page can be generated from.

4.2.2 Implement

Ideally, a project encourages a group of people to implement new features, fix bugs, etc. Therefore, the project needs to have a contribution guide, especially for those who want to get started. Generally, building a development community is a major requirement for a successful project in the long run. In addition, it must be ensured that all committers are aware of IP issues when using third-party libraries which have to be tracked. However, during development there are some restrictions regarding the use of software for development support. For example, all Eclipse projects have to use Bugzilla to track bugs and features. As a version control system, SVN or Git should be used. Although Maven is recommended as build tool, others can be used as well. Though, projects have to move to the Eclipse provided Hudson build servers. However, we will not go into detail at this point since tooling must be clarified for each project separately.

4.2.3 Produce Milestone

Projects are encouraged to use an agile development process including regular milestones. According to Eclipse, former projects following the waterfall model had been less successful. Although the time period between the milestones can vary, major projects (e.g. the Luna project: Free IDE version of Eclipse) have shown that a six week milestone schedule is reasonable. With each milestone, a complete software development lifecycle is conducted: Design, code test, review, integrate and distribute. It is also recommended to establish a predictable, reliable milestone schedule without changing major points during the process. The advantage of short-termed milestone builds are that developers get a fast feedback and the community is kept alive because of regular new features.

4.2.4 Release

In fact, *Eclipse Foundation* is geared to the release process of the Apache Software Foundation where detailed information can be found. A release is a version of the project which is distributed to the outside. If users are directed on a website to download a build, this version must have been released. Any links to nightly or integration builds should not be made public in wikis, blogs, etc. However, milestones and release candidates are exceptions as they can be made available to the community including the mandatory caveat that these versions are not official. Official releases are the only downloads allowed to be labeled with x.y, e.g. 0.5, 1.0, 2.3, etc. The time period during releases is not fixed.

All official releases must undergo a prior review process which has the following four main purposes:
(i) Summarizing the accomplishments of the release (e.g. major features, controversial extensions, etc.),
(ii) checking that no IP Policies have been violated,

(iii) highlighting any remaining issues of quality, security, usability architectural etc. and (iv) finally to verify that the principles and purposes of Eclipse are satisfied.

The process starts with assembling review documentation and the IP Log. For the latter, it is useful to use tools such as Project Downloads Scanner to confirm that only approved third-party code is in the project or the Bugzilla Contribution Review Tool. After that, the IP Log should be submitted to the IP Team and the review documentation must be approved by the PMC. Finally, the release can be finalized.

4.2.5 Release Checklist

The following list, based on [Eclipse Foundation, 2014b], gives an overview of important tasks which have to be checked prior to releasing.

▻ Project Website up-to-date (including Contribution Guide)?

▻ Project meta-data is up-to-date (e.g. description, release metadata, project plan)?

▻ Check if code and builds are properly managed

- project source code available in Eclipse repository
- contributing files
- at least one milestone build
- builds hosted on eclipse.org
- naming conventions and version numbering rules followed
- bundle manifests correct
- bundles contain an about.html file for legal issues

▻ Naming conventions and version numbering followed?

▻ Review security-related bugs

▻ Download files include the required legal documentation?

▻ IP Logs submitted for review by the Eclipse IP Team?

▻ PMC approval obtained?

▻ Review documentation (correct copyright, EPL notice, URLs of the project, etc.)

Chapter 5

Case Study: inspectIT going Open Source

This Chapter is devoted to document the most important software engineering processes of *inspectIT*. Since the development of *inspectIT* does not follow a common software process model such as SCRUM, a higher level perspective is needed to define the software processes. For this purpose, the following sections are organized according to the reference model of the *Unified Process for Education (UPEDU)* according to Robillard et al. [2002]. The main disciplines share a common structure as illustrated in Figure 5.1.

5.1 Software Engineering Processes of inspectIT

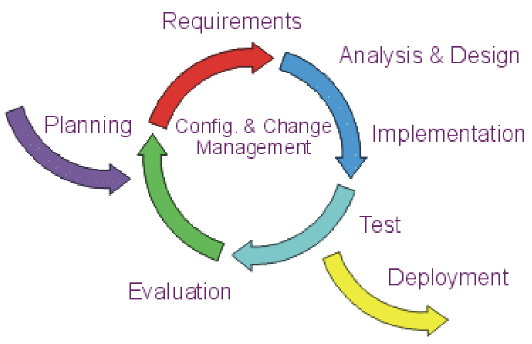


Figure 5.1. The set of disciplines in a typical software engineering process Robillard et al. [2002].

The model reflects the iterative lifecycle processes of *inspectIT* quite appropriately as it is done in practice. However, the model also covers the development of a completely new software, which is why the following sections do not include all phases of the model. For example, the planning phase is essential when beginning a new software project but it is not part of the iteration and therefore left out. The first process to analyze is the requirements analysis of *inspectIT*, in terms of new features. Thereafter, the development phase will be reflected before the testing processes are presented. In the context of *inspectIT*, the evaluation is done in a special way by integration managers. That is why we renamed this phase into integration and build processes. The deployment phase will be considered as the new version releases. It should be noted that currently only a few persons are actively involved in the development processes of *inspectIT*. The following paragraphs are mainly based on the internal documentation provided by NovaTec Consulting GmbH.

5.1.1 Requirements

The current *inspectIT* development is requirement-driven as this is the first point of each activity. The management of requirements is handled via the internal Jira ticketing system. A "new Feature" ticket contains information about what should be done and the reasons why to do so. Usually, requirements are refined into several more concrete tasks. Of course, not only new features constitute to requirements but also a bug/defects or other (internal) tasks are treated as new requirements. Each ticket is assigned to a developer and an estimation for realizing, testing and documenting must be given. The owner of the requirement should act as a customer and should be available if any questions arise. Often, new requirements arise due to practical experience with the tool.

Possible improvement: Currently, it is not obvious which requirements should be realized and which not. Especially, it would be helpful for further planning, if something like a long-term road map for *inspectIT* was created. It contains all planned features, described very roughly, on a timeline. Then, implementing new features would be transparent to all participants.

5.1.2 Development

In order to correctly set up the development environment for new *inspectIT* developers, a guideline was created to install and configure all the tools necessary for implementing new code (e.g., Ivy, PMD, TestNG, Findbugs, Checkstyle, etc.). In general, all developers get several tickets assigned by the Jira system. There are multiple ticket statuses in Jira such as open, in progress, integration needed, resolved, closed to just name a few. Each ticket, which can also be a reported bug, needs to be classified based on a specific classification (blocker, critical, minor, trivial). At least two developers must agree on the severity of the bug. After developers think they accomplished a ticket, they can request the integration into the master branch (See Chapter 5.1.4).

5.1.3 Testing

inspectIT undergoes several types of tests which are described in the following.

Unit tests are designed to test the functionality of implemented methods and core classes. Relevant incoming and outgoing data, that are needed for a unit test, are usually mocked. Developers must write appropriate unit tests for their code in order to request a successful merge into the master branch (See Chapter 5.1.4).

Component/Integration tests are written for components (Agent, CMR, GUI) using the Spring framework. These tests are more abstract than unit tests as the interaction with multiple modules are very important to test. It is necessary since some functionality (e.g. bytecode instrumentation of an application) must be tested according to its outcome.

Explorative Testing is conducted typically one complete day before each release. There are defined steps which are actually some kind of use cases, executed manually. The use cases cover major capabilities of each of the components. Explorative testing should ensure that no critical bugs were found in the release version. However, minor bugs which were detecting during testing are reported and documented on Jira for the next release version.

All automated tests are integrated into the build process. However, there is a load test which is executed every weekend. Therefore, a sample application (DVD store) is monitored by *inspectIT* in order to analyze how *inspectIT* behaves if the application experiences heavy load.

Possible improvement: For now, there are no fully automated performance benchmarks to asses the performance overhead of *inspectIT* itself. At some critical points, some rudimentary performance experiments are built, e.g., the performance overhead of the exception sensor which revealed that tracing the stack trace does reduce performance significantly. Though, these tests are just to get a feeling about the magnitude. A generic way to measure performance overhead of *inspectIT* in a fully automated way would be interesting. Furthermore, an accurately measured performance overhead (which should be low) are hard facts for customers in order to increase their confidence in this tool.

5.1.4 Integration

The integration process of *inspectIT* has the goal to only integrate high quality code into the master branch. First, the *inspectIT* developer starts the process by creating a new branch and cloning the current version of *inspectIT* into the new repository (Gitorious is used here). In order to fulfill one specific task, represented by a Jira ticket, the developer writes some code. After he/she finished, the code has to be pushed into his/her remote branch, eventually. Next the processed Jira ticket has to be set to "integration needed" which is essentially a merge request into the master branch. Jira notifies the role called "integration manager" who is permitted to merge the change into the master branch. Currently, only three *inspectIT* members have this permission. The integration manager reviews the merge request, focusing on code quality (static code analysis), sufficient testing, etc. In practice, is an interaction between the developer and the integration manager on how to accomplish this task the best way and in high quality. After all, if both think that the change is mature enough, the new code is pushed into the master branch. If the Jenkins build was successful (e.g., no test failures), the integration manager closes the merge request and the ticket.

5.1.5 Build Process

Each of the three *inspectIT* components has its own build process. Ivy is used for resolving external dependencies. Continuous integration is done with Jenkins. Code quality tools such as FindBugs, Checkstyle and PMD are executed when running a build/release job and reports are generated.

Essentially, the two important jobs are the integration job and the release job. On the one hand the integration job is executed automatically after a commit into the master branch of *inspectIT* (only permitted for integration managers) and verifies that no problems occur during the build process. On the other hand, there is a release job started manually when necessary. The complete build will be issued and the release artifacts are uploaded to the FTP server.

5.1.6 Configuration and Change Management

Changes to requirements are created by the project lead. An original ticket can be enriched with an additional ticket e.g., as an enhancement or providing additional information regarding the ticket. Therefore, the initial Jira ticket has a history list of all changes to it. However, proposed changes can also be rejected if they are not compatible with the desired goal. The primary goal of configuration management is to ensure the control of changes made to hardware, software, firmware, documentation, test, test fixtures, and test documentation throughout the life cycle of the software engineering process. For this purpose, the *inspectIT* team uses three systems:
(i) Jira for traceable requirements, tasks, bugs and change requests,
(ii) Confluence for the internal and user documentation and
(iii) Git for the source code version control.

These system ensure that all software artifacts can be traced at various points in time.

5.1.7 Release Process

For the *inspectIT* team, a fully automated release process is very important. All software bundles are created automatically whereas some organizational tasks (e.g., changing the download link on the *inspectIT* homepage or generating release notes to confluence) are done with manual interference. Furthermore, the time periods between the public releases is not defined: As the development is ongoing, small improvements or new features are integrated regularly. First, beta versions are released when major bugs are fixed or the last release was four months ago. However, if the team thinks that some major changes are done justifying a main release, a new version with an incremented number (e.g. from 1.6 to 1.7) is released. A major release has some requirements: sufficient testing, critical or blocking bugs have been fixed and all planned features for this version were realized. All releases are started from the Jenkins server. Depending on the release type, specific tasks have to be executed. The following list gives a rough overview of the tasks:

- Build and test
- Upload artifacts to FTP
- Update homepage links
- Add link to release notes on homepage
- Create release notes
- Move confluence spaces (only if major release)
- Integrate news about new release on homepage
- Finalize Jira version

5.2 Application of the Eclipse Foundation's Requirements on inspectIT

In this section we give some examples what had to be changed or adapted if NovaTec Consulting GmbH had decided to become an Eclipse project. Since NovaTec is not publishing *inspectIT* within the Eclipse cosmos, we are not going too much into detail. With the short case study we provide an opportunity for other projects to see what impact the decision to publish as Eclipse project might have. This section is based on Chapter 3 where the requirements becoming part of the *Eclipse Foundation* are discussed in general and on Chapter 5 with the description of the current processes of *inspectIT*.

One task which is necessary to accomplish before starting the joining process is to use Bugzilla as issue tracking platform. Right now Jira is used in the *inspectIT* processes for that purpose. As a result, already tracked information have to be transferred into Bugzilla as well as developers and other users of that system need to get trained in using Bugzilla.

Another point NovaTec Consulting GmbH had to change before their product is able to become an Eclipse project is their building process. Right now *inspectIT* is build with Jenkins. The *Eclipse Foundation* requested that all products with are an Eclipse project have to be build with the Eclipse infrastructure with includes a Hudson server. Fortunately, a transformation could be done with a low amount of working hours.

One main restriction for a project which is becoming an Eclipse project is that no code, or libraries can be used if they are published under license which does not conform with the EPL like for example (L)GPL. Meeting this requirement might cost the most time. To accomplish that task, every library or code that is not development within the project has to be listed in the tool IPZilla. If some code is published under a license which is not conform with EPL it has to replaced with some code which is conform. That could cause some trouble because some parts of the project might have to be changed. The IP team of Eclipse checks the whole code itself so it is not possible to hide not conforming license from them. NovaTec Consulting GmbH would have checked their code, if they would have agreed to publish their APM tool *inspectIT* as an Eclipse project.

Also the developers NovaTec Consulting GmbH which are mainly developing for *inspectIT* would have some new tasks in their daily work. They would have to do some additional work for the open source community which is shown in Figure 5.2. Besides the development of the tool, they also should do some Eclipse related work like administrating forums of the project or presenting the project on some Eclipse Conferences.

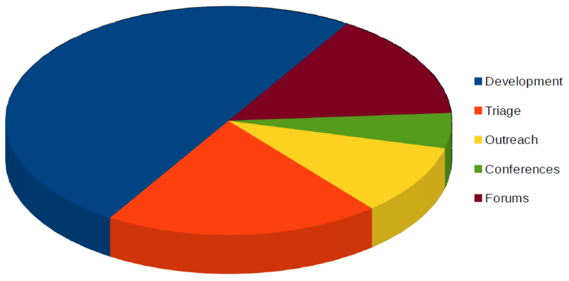


Figure 5.2. Tasks of a developer of NovaTec Consulting GmbH after *inspectIT* become an Eclipse project. Beaton et al. [2014]

Beside the technical challenges NovaTec Consulting GmbH would have to transfer the copyrights of *inspectIT* to the *Eclipse Foundation* . Even if the expert knowledge about *inspectIT* stays within NovaTec Consulting GmbH other companies could take *inspectIT*, make some changes and could not publishing this changes under the open source license again. NovaTec Consulting GmbH could lose control about the future development of *inspectIT*. The restriction of the *Eclipse Foundation* to give up the rights about the project and the name of *inspectIT* with no possibility to undo the decision to publish *inspectIT* as Eclipse project was one of the argument for NovaTec Consulting GmbH not to publish *inspectIT* as an Eclipse project.

5.2.1 Case Study: Jubula becoming an Eclipse project

Another case study which describes how the test tool *Jubula* migrated to the *Eclipse Foundation* is published in the Eclipse Magazine divided into four parts. *Jubula* consists of main features provided in the commercial version of the testing tool *GUIdancer*. So Bredex GmbH on the one hand owned the Eclipse project *Jubula* and on the other hand the commercial tool *GUIdancer* with some extended features compared to *Jubula*. After the positive feedback and success of the open source tool all functionality of *GUIdancer* were integrated into *Jubula*. Similar to the *inspectIT* project, a huge amount of code existed at the beginning of publishing *Jubula* as an Eclipse project. Learning from the experience of

Jubula would be certainly helpful if *inspectIT* or a similar projects should start becoming an Eclipse project.

The first part [Loerke, 2011a] summarizes Bredex's experience about the starting process of publishing *Jubula* as an Eclipse project. The main reason why *Jubula* became an Eclipse project is that Bredex wanted to concentrate their businesses case on giving consulting, enhancement development and offering training courses. During the time they were in the process of publishing *Jubula* as an Eclipse project, they received great help from the mentor program of the *Eclipse Foundation* .

In the second section [Loerke, 2011b] their experiences with the IP regulations of the *Eclipse Foundation* are documented. The whole IP checking by the *Eclipse Foundation* is documented as well as the changes needed to be done to publishing *Jubula* as an Eclipse project. They used *ParallelIP* for that purpose. After a review by the *Eclipse Foundation* they had about 20 minor change request which were manly focused on legal issues like the request to change some comments, missing EPL headers or methods- and class names which are similar to other open source software projects.

The third part [Ford-Reitz, 2011] shows that Bredex had to change a component of their software project before they were allowed to publish *Jubula* as an Eclipse project. In *GUIdancer* the basis of *Jabula Hibernate* was used as an Object-relational-Mapping-(ORM-)framework. Since *Hibernate* is published under the LGPL, the framework is not allowed to be used within an Eclipse project. As a consequence Bredex had to replace that framework. In order to solve this problem the ORM-framework *EclipseLink* was found in the Eclipse Cosmos. After some migration steps they were able to replace *Hibernate* with *EclipseLink*.

In the final part [Tiede, 2012] the tasks of a developer occurring if his/her project becomes an Eclipse project are described. The refactoring process was mentioned since they took code of their commercial tool *GUIdancer* for *Jubula*. Furthermore, the author gives a feedback about the revision control tool git. At Bredex *Subversion* was used before, so they had to convert according to *Eclipse Foundation* restriction to git. The converting process was not a problem since they did not transfer old entries from *Subversion* to git. But according to the author understanding the whole functionality of git is a long process. Another point they had to adapt according to the regulation of the *Eclipse Foundation* is the development process. The development process they used before was focused on company internal development and was not adaptable to an open source development process. They also had to outsource the entire building infrastructure to the *Eclipse Foundation* . After some experiments *Jubula* builds every day under a combination of Maven3 and Tycho in a Hudson-CI-Infrastructure.

As a conclusion they mentioned that the whole process was not always easy to go or straight forward but they were successful and a high amount of requests and users of

Jubula showed them that they did the right thing. One important task after becoming an Eclipse project is to update the project continuously with new features and updates.

Chapter 6

Conclusion

In this process analysis, the conditions for a software joining the *Eclipse Foundation* and therefore the Eclipse community are analyzed. For this purpose, the documentation of the EPL and of the *Eclipse Foundation* was studied and a step by step guidance was created to give an overview about which aspects in joining the *Eclipse Foundation* have to be considered. As a case study the development process of *inspectIT* was analyzed in order to prepare *inspectIT* for joining the *Eclipse Foundation* .

One goal of this process analysis was to evaluate theoretically which aspects of the APM tool *inspectIT* need to be changed to join the *Eclipse Foundation* . For that purpose the software planning, development and maintenance process of *inspectIT* was documented. By comparing the current status of *inspectIT*'s processes with the requirements of the *Eclipse Foundation* we identify which steps would have been required to join the *Eclipse Foundation* . To have experience of a project which successfully became an Eclipse project we analyzed a case study of the project Jubula and its way to an become part of the Eclipse Cosmos.

In summary, it would have been possible for NovaTec Consulting GmbH to publish *inspectIT* under EPL. Although some tools and process steps in development software had to be changed before. The decision of NovaTec not to publish *inspectIT* under the EPL is mainly based on legal issues. One main problem was that after publishing a project as Eclipse project, the project owner has to transfer rights of the project towards the *Eclipse Foundation* . In future *inspectIT* will probably be published under an other open source license.

The technical requirements and issues in joining the *Eclipse Foundation* were analyzed in this process analysis. The legal aspect, which is as important as the technical aspect, is covered on a high level point of view. For further work this could be focused at legal issues in analyzing the EPL and *Eclipse Foundation* .

At the end of our documentation we want to give a personal feedback about the EPL. In our own opinion, the license is a weak copy-left license meaning that one is not required to use the same license for a derived work. So one can take EPL licensed code, include it in his own work, and distribute the result under a non-free, commercial license. Therefore, it is theoretically possible that another company could take the source code, add closed-source features and sell the product. We found this a downside of this license model. The benefits are that EPL is non-restrictive and developer-friendly. Except of the combination of different license models, a developer has a lot of freedom what to do with the code although they can't make claims because they have no copyright of their own code they contributed if the project become an Eclipse project.

Our personal experience regarding the research towards *Eclipse Foundation* and its processes was, that although there is a lot of documentation material that probably covers all aspects in detail, the biggest problem was to get a proper overview over all this material. The biggest hindrance was the huge amount of cross-linked resources. This lead to partially overlapping sources towards one topic.

During our project our team received great assistance from the side of our industry partner NovaTec Consulting GmbH especially by Ivan Senic who assisted us even after the decision not to converting *inspectIT* into an Eclipse project was made and on the other side by André van Hoorn from the University of Stuttgart.

Bibliography

[Beaton et al. 2014] W. Beaton, J. Campbell, and D. Roy. Eclipse Committer Bootcamp. Online, 05 2014. URL http://wiki.eclipse.org/images/b/be/EclipseCommitterBootcampOpenMay2014.pdf. Last accessed: 02.12.2014.

[Behr and Tyll 2003] T. Behr and T. Tyll. Prozessanalyse. *Geschäftsprozesse–Organisatorische Gestaltung. Online-Lehrbuch, Universität Erlangen*, 2003.

[Eclipse Foundation] Eclipse Foundation. Eclipse Public License - v 1.0. URL http://www.eclipse.org/org/documents/epl-v10.php. Last accessed: 14.12.2014.

[Eclipse Foundation 2003] Eclipse Foundation. Bylaws of Eclipse Foundation, INC., 2003. URL http://www.eclipse.org/org/documents/Eclipse%20BYLAWS%202003_11_10%20Final.pdf. Last accessed: 08.12.2014.

[Eclipse Foundation 2011] Eclipse Foundation. Eclipse Intellectual Property Policy, 2011. URL http://www.eclipse.org/org/documents/Eclipse_IP_Policy.pdf. Last accessed: 08.12.2014.

[Eclipse Foundation 2013] Eclipse Foundation. Eclipse Foundation Contributor License Agreement, 06 2013. URL http://www.eclipse.org/legal/CLA.php. Last accessed: 14.12.2014.

[Eclipse Foundation 2014a] Eclipse Foundation. Eclipse Development Process 2014. Online, 2014a. URL http://www.eclipse.org/projects/dev_process/development_process_2014/development_process_2014.pdf. Last accessed: 02.12.2014.

[Eclipse Foundation 2014b] Eclipse Foundation. Eclipse Development Process Release Management 2014. Online, 2014b. URL https://wiki.eclipse.org/Development_Resources/HOWTO/Release_Reviews#Checklist. Last accessed: 02.12.2014.

[Eclipse Foundation Wiki] Eclipse Foundation Wiki. Eclipse Wiki - Development Process. Online. URL http://wiki.eclipse.org/Development_Resources. Last accessed: 02.12.2014.

[Ford-Reitz 2011] Z. Ford-Reitz. Waking up to EclipseLink. *Eclipse Magazin*, (6), 2011.

[Loerke 2011a] A. Loerke. Aufzucht und Pflege eines Eclipse Projekts. *Eclipse Magazin*, (4), 2011a.

[Loerke 2011b] A. Loerke. Intellectual Property: Wozu der Aufwand? *Eclipse Magazin*, (5), 2011b.

[Martin Hörst 2011] A. O.-A. Martin Hörst. A systematic review of research on open source software in commercial software product development. *Information and Software Technology*, 53(6):616–624, June 2011.

[Robillard et al. 2002] P. N. Robillard, P. Kruchten, and P. d'Astous. *Software Engineering Using the Upedu*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[Tiede 2012] M. Tiede. Open Sourcing and Release Engineering @ Eclipse.org. *Eclipse Magazin*, (1), 2012.

Declaration

We hereby declare that the work presented in this thesis is entirely our own. We did not use any other sources and references than the listed ones. We have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. We have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

Place, Date	Thomas Düllmann
Place, Date	Tobias Rudolph
Place, Date	Anton Scherer