Institute of Software Engineering
Software Quality and Architecture

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Eclipse Installation Manager - A Tool To Manage Eclipse Product Installations

Alexander Schindler

Course of Study:	Softwaretechnik
Examiner:	Prof. Dr.-Ing. Steffen Becker
Supervisor:	Prof. Dr.-Ing. Steffen Becker
Company Affiliation:	Kirschners GmbH
Commenced:	May 17, 2023
Completed:	November 17, 2023

Abstract

Context. Developing and evaluating a software to help developers with a specific problem which exists in the current workflow for Eclipse developers.
Problem. Managing Eclipse Integrated Development Environments can become problematic and a chore to the developer when handling many of them for a lot of projects over an extended period of time.
Objective. Write an application that integrates into the daily development process and evaluate the result.
Method. The application is written in Java and based on the Eclipse Oomph Project and Java SWT. The impact of the tool is measured using a guided peer study, where participants were asked to complete a list of relevant tasks and rate the subjective effectiveness and impact of the Eclipse Installation Manager on their workflow.
Result. The results indicate that the Eclipse Installation Manager has a positive impact on the interaction between developers and their development environments. However, the application itself needs to be more refined when it comes to the consistency and the user interface.
Conclusion. The concept of the Eclipse Installation Manager shows great potential and is validated by users who liked the idea and knew the problem from their own experiences. The solution to the problem of managing workspaces and installations is a little more complex and multiple approaches are possible. Future work focuses mostly on improvements of the implementation, but additional studies could reinforce the significance of the problem and concept and maybe evaluate the impact in an objective manner.

Kurzfassung

Ein weit verbreitetes Entwicklungswerkzeug ist die Eclipse Entwicklungsumgebung. Aufgrund der modularen Natur der Eclipse IDE kann sie für verschiedene Projekte und Programmiersprachen verwendet werden. Entwickler haben auf ihrem Computer oft mehrere verschiedene Installationen der Entwicklungsumgebung, was schnell unübersichtlich und schwer verwaltet werden kann. Das Ziel ist es daher ein Tool zu entwickeln, das den Entwickler dabei unterstützt schnell und effektiv die richtige Entwicklungsumgebung zu starten, und die vorhandenen Umgebungen verwalten. Der Eclipse Installation Manager adressiert das Problem, indem eine kompakte, verständliche Übersicht und Verwaltungsoberfläche bietet. Die Bachelorarbeit untersucht den Einfluss der erstellten Software auf den Entwicklungsprozess und Teilnehmende der Studie sollen bewerten wie sehr der Eclipse Installation Manager im täglichen Arbeitsablauf unterstützt. Zusätzlich wird die Nutzererfahrung des Programms untersucht. Die Ergebnisse zeigen, dass das Problem unter Entwicklern bekannt ist, jedoch keine bisherige zufriedenstellende Lösung existiert. Der Einfluss des Ansatzes ist größer, je eher ein Entwickler bereits bestehende Strukturen wie den Eclipse Installer nutzt. Teilnehmende der Studie empfanden die Anwendung als hilfreich, auch wenn es einige Bedenken bezüglich der spezifischen Implementierung gab. Hier besteht noch Verbesserungsbedarf und dementsprechend bezieht sich die zukünftige Arbeit hauptsächlich auf Verbesserungen der Implementierung. Zusätzlich dazu könnten weitere Studien, welche den Einfluss der Anwendung oder des Konzepts objektiv betrachten, mehr Aufschluss darüber liefern, wie viel Zeit ein Entwickler einspart.

Contents

List of Figures

List of Tables

1 Motivation

The usage and development of software has changed a lot over the years. Developers today are often involved in multiple projects with varying requirements when it comes to the necessary tools. For example a developer who develops a website which uses a JavaScript framework for the frontend and might have to make use of a Spring Boot Framework based on Java as backend. Both tasks can be fulfilled with the Eclipse Integraded Development Environment (IDE), but in order to do so the developer would need different packages of the Eclipse IDE. This leads to a growing number of Eclipse IDE installations and discloses a question: How does a developer manage these installations and their accompanying workspaces? At the moment there is no clear answer and every developer does it on their own, usually using the operating systems file manager. This process can be cumbersome, especially when outdated installations from old projects have to be dealt with. This is the first problem this thesis aims to address: Create a piece of software, that allows to start, modify and remove installations of the Eclipse IDE, ultimately simplifying the workflow of developers. To reach that goal the proposed software needs to be customizable in a way that allows a developer to identify installations and workspaces.
The second use case revolves around the Palladio Bench. In short, the Palladio Simulator allows developers to simulate software architectures to quantitatively analyze them. Ultimately, the Palladio Simulator's goal is to improve software quality and reduce trial-and-error-cycles [Palladio]. Palladio comes in many different configurations to address different use cases itself, like Reverse Engineering or Quality Analysis. This leads to an analogous problem as already discussed: A user might want to have multiple different product installations available, which can get hard to manage. As the goal is to manage all Eclipse Applications, the Palladio Bench needs to be installable with the official Eclipse Installer, since that tool provides necessary metadata about installations and workspaces. Enabling that is the second contribution of this bachelors thesis.
The Eclipse Installation Manager is the resulting tool and ideally would be a Plug-In, which can be combined together with the Eclipse Installer, including its functionality and extending the installers functionalities naturally.
The development during the thesis utilizes an agile approach to be able to adjust the direction of development and iterate on the steps already taken. This approach includes the requirements discovery and engineering for the proposed software, adaptive planning and continual improvement. The resulting thesis evaluates the efficacy of the Eclipse Installation Manager in the context given, while also discussing the software design and what can be improved. Hence, the main research question aims to investigate if the implementation and introduction of the proposed software positively affect the way developers interact with and manage their development environment. Additionally, it was evaluated how satisfied the users were with the functionalities and user experience of the Eclipse Installation Manager.

Thesis Structure

Chapter 1 – Motivation: This chapter introduces the main problem addressed by the following thesis and provides a first idea of how to solve it.

Chapter 2 – Foundations and Related Work: Here, the technologies and software pieces that are necessary to understand this proposal will be explained. In addition, previous research investigating similar issues has been analyzed and potential other solutions will be assessed in this chapter.

Chapter 3 – Requirements Engineering: This chapter explains requirements engineering basics and discusses what has been done for the requirements of the Eclipse Installation Manager.

Chapter 4 – Development: Based on the elicited requirements the software is implemented. This section provides an architectural overview and explains the design of the software, as well as challenges during the development process.

Chapter 5 – Evaluation: To evaluate the impact of the Eclipse Installation Manager a guided peer study was designed and executed. This chapter includes the study structure, results and limitations.

Chapter 6 – Conclusion: The conclusion serves as a summary of the thesis including short paragraphs about the concept, limitations and work which could build upon this thesis.

2 Foundations and Related Work

This chapter introduces the foundations needed to understand the **Eclipse Installation Manager** (EIM), its aspects and what it tries to achieve. This will begin with theoretical basis followed by the tools used. The first section clarifies the terminology used by the Eclipse Foundation inside the Eclipse Platform. Following that, the principles of the Open Services Gateway initiative (OSGi) are explained, as these are a substantial part of the Eclipse Platform. A short introduction into the Eclipse Rich Client Platform is then followed by the Eclipse Modeling Framework (EMF).
The introduction of the tools begins with the Eclipse Installer and is followed by bndtools afterwards.

2.1 Foundations

The following section explains the terminology and concepts used in Eclipse and OSGi.

2.1.1 Terminology

The Eclipse Platform started as a framework for building *integrated development environments* (IDEs) and developed into a platform where its architecture supports the development of any application while only using a subset of the Eclipse Platforms components [EMF09]. At the core of this architecture are Plug-Ins. In the Eclipse World a *Plug-In* is a component and refers to a basic unit of function. The Eclipse Platform can be extended in functionality with Plug-Ins and is composed of Plug-Ins itself. The *Plug-In architecture* builds upon the OSGi Service Platform, which provides a component frame and enables installation of components at runtime [EMF09]. The OSGi specification refers to *bundles* as a unit of modularization. The terms Plug-In and bundle can be used interchangeably [VM13]. A bundle contains everything that is needed to run the component, more specifically it includes two manifest files, that identify the bundle, provide dependency information and declare available extension points as well as used extensions. Another component of the Eclipse Platform is the *Platform UI*, which includes toolkits to generate platform-independent user interfaces based in widgets. In addition to that it provides a framework to customize available actions. Generally, the main UI window is referred to as *workbench*.

2.1.2 OSGi

The **Open Services Gateway initiative** is a set of specifications for a Java framework, which describe a dynamic component model. Since 2020 the initiative is part of the Eclipse Foundation as *OSGi working group* [OBL]. The OSGi specification refers to plug-ins as bundles. An application consists of multiple components which can be referred to as a packaged bundle. Bundles have an

independent lifecycle, which will be discussed later in more detail. The principles behind OSGi help to understand what the specifications try to achieve. These principles have been written down in [OSGi3]. A concise overview and explanation is given to connect later design decisions to the architecture of the specification.

Principle 1: Versioning Uniquely identifies a revision and, when used in semantic versioning, also contains information about updates and breaking changes

Principle 2: Prepare for change As changes will occur invariably the code has to be optimized accordingly. This has to be done continuously to keep up with changing software requirements.

Principle 3: Minimize the cost of change When changes happen the cost (and impact) should be as small as possible.

The first principle is addressed in the technical whitepaper "Semantic Versioning" [SemVer] and specifies major and minor versions, a bugfix and qualifier, with the syntax `MAJOR.MINOR.BUGFIX.QUALIFIER`. When changes occur, which break the backwards compatibility, an increase to the major version is necessary. This means that an artifact compiling against the older version will not be compatible with the newer revision. The minor version increases when a new revision breaks the compatibility for providers of the API, but not for consumers. Bugfix and qualifier indicators have no effect on backwards compatibility.
The semantic versioning whitepaper also specifies how a consumer or provider of an API has to pay attention to the package import versions, if semantic versioning is used. As a result a provider would import version 1.5 but explicitly not version 1.6, not regarding bugfixes or qualifiers using `[1.5,1.6)` and a consumer can import the version `[1.5,2)` with no further considerations. Importing versions down to the qualifiers does not necessarily make sense because `QUALIFIER` and `BUGFIX` releases , if done correctly, are not supposed to break anything with your code. Hence when specifying the range your artifact imports, `MAJOR.MINOR` is perfectly sufficient.
The second principle is a bit more complex to solve, as changes can happen quickly on top of already existing work. The solution proposed by the OSGi Alliance is Modularization. Modularization is the decomposition of a system into independent, but connected parts, where their aggregation forms the system. Proper Modularization can simplify a system, but, if done wrong, will make a system more complex. Modularization can have multiple advantages for the developer:

- A module can decide what parts it makes visible (exports) to other modules. This simplifies the system, as it is reducing the amount of parts in a system.
- Changing code in a module is easier, as the scope of the changes is limited, if the changes do not affect the exported parts of the module.
- Respecting a modules boundaries creates a local scope which affects the scope of changes in merged packages.

It is important to emphasize, that developers often refer OSGi to the reference implementation of the OSGi specification. The specification sets the requirements any implementation of the specification has to fulfill, but does not define how to do it. The most important aspects of the specification for this project are the *OSGi framework* and the *Declarative Services Specification*. Without going into too much detail, the parts a briefly explained in the following paragraph.
The OSGi framework consists of 4 layers: a security layer, a module layer, a life cycle layer and a service layer. The security layer is optional and managed by the life cycle layer. The goal here is to

have an infrastructure for applications in fine-grained controlled environments. The module layer standardizes modularization for Java and specifies what a bundle is, what it needs to include and what it exports in the form of *capabilities*. Bundles can provide a capability and require others, forming the dependency tree. Additionally, it specifies the resolution of any bundles external dependencies [OSGi1].

The life cycle layer uses the module and security layer and provides an interface to control the life cycle operation of bundles. Frameworks as well as bundles in a framework have states, as shown in Figure 2.1 and follow a life cycle: After installing a bundle into a framework, the framework tries to resolve necessary external dependencies, as specified in the module layer. A resolved bundle can be started, stopped and activated.

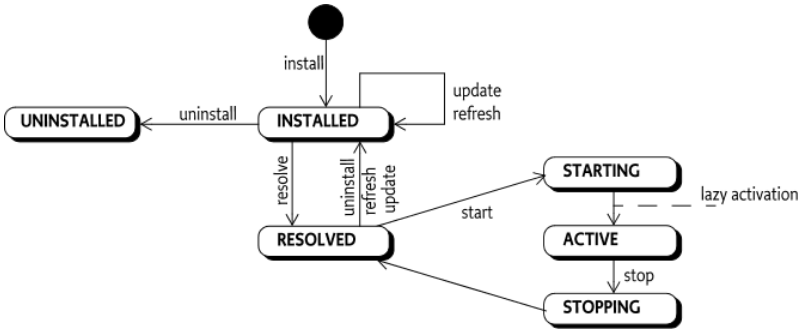


Figure 2.1: Available bundle states for OSGi bundles in an OSGi framework including the transition between states [OSGi1]

The service layer is the last one, where bundles can public, bind or find services. A service is registered under a Java interface with the service registry and can be found by searching bundles. This enables a dynamic exchange of the bundle registering the service, without any need to modify the bundle which binds the service. The concept is shown in Figure 2.2 [OSGi1].

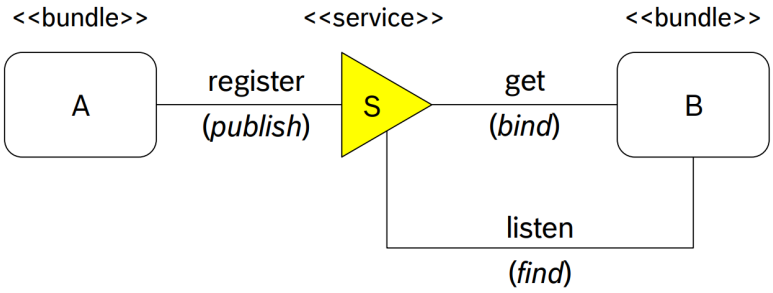


Figure 2.2: Depiction of a bundle A which registers a service and provides the service component and another bundle B which can listen to a service and bind it as a consumer [PK].

This specification of the OSGi *service model* is used with *declarative services*. Declarative services use the *service component model*, where a component annotation inside a Java class specifies publishing, searching and binding registered services, effectively increasing the productivity of the developer by lifting off responsibility by automating the service process.

The Eclipse Installation Manager project uses OSGi declarative service specification because it enables modularity and divides responsibility. Many Eclipse projects use OSGi themselves and Eclipse Plug-Ins are OSGi bundles. Developing a bundle with the potential goal of integrating the Eclipse Installation Manager into the official Eclipse Installer is therefore necessary. Additionally declarative services allow the developer to dynamically load components only if they are needed. Overall the mentioned benefits make the usage of OSGi declarative services for developing to Eclipse Installation Manager very likely.

2.1.3 Eclipse RCP

While the Eclipse Platform has been used to develop rich client applications, the official proposal to simplify the use of the Eclipse Platform as a rich client platform was proposed as early as 2003 with the reasoning that the platform already provides many components not specific to IDEs [RCP]. The functionality so far provided and tested by the Eclipse Platform, such as window-based GUI, plug-in system or update system, can be easily reused in rich client applications. The Eclipse Rich Client Platform is often described as "a minimal set of plug-ins needed to build a rich client application" ([EMF09], p. 9).
The Eclipse Installation Manager is planned to be implemented as a lightweight rich client application based on the Eclipse RCP. A thin client (that is, a web application) is not suitable, as installations and related installation information are stored locally. The Eclipse Rich Client Platform is planned to be used as basis on which the Eclipse Installation Manager should be deployed which enables a user to install the Eclipse Installation Manager using the official Eclipse Installer.

2.1.4 Eclipse Modeling Framework (EMF)

Before introducing the Eclipse Modeling Framework the notion of a model should be clarified. Software Engineering knows two types of models, descriptive and prescriptive models. The first are used to describe an object, so the model is based on a real world object. For example, a car can be modeled descriptively by a simple UML diagram, where classes like Car, Engine or Wheel container properties of the real world objects. In contrast to that prescriptive models serve as basis for a realization as an object or software. Modeling is the process to get a model from descriptions for use cases or requirements. The model describes what an application is supposed to do and what it is composed of. EMF is a framework to ease the use of model-driven software development (MDSD) by enabling developers to generate an EMF-model from Java code, an UML model or a XML-based data model, and vice versa. So effectively a developer can create a prescriptive model for a software and EMF allows the developer to generate a code implementation using Java.
The EMF-model is a simple model of the data, classes and the application and is based on the class modeling aspect of UML. Compared to standard UML, which includes many more diagrams, like the activity diagram or state diagram, the reduction simplifies the usage of EMF, and the code generation based on these UML diagrams can be very complex [EMF09].
The Ecore Model is an EMF-model that represents EMF-models, so it is its own meta-model. Figure 2.3 shows a simplified version of the Ecore meta-model, which is enough for a simple example. The full Ecore meta-model can be seen in [CEM]. The example includes ideas how the Eclipse Installation Manager could be modelled, but the following list only serves the example here.

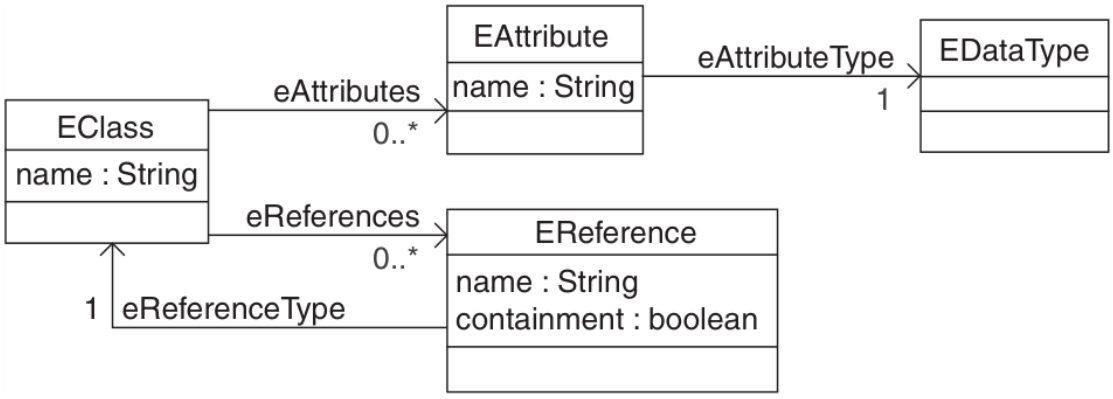


Figure 2.3: Part of the Ecore Meta-Model. It is used as a basis for a simple example regarding the modeling of the Eclipse Installation Manager [EMF09].

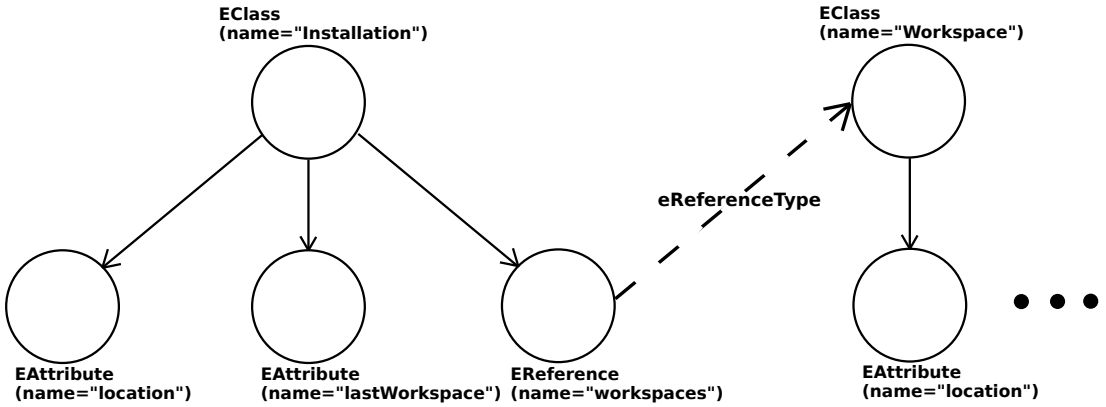


Figure 2.4: Exemplary instantiated Ecore meta-model that serves as an example in the realm of the Eclipse Installation Manager, to clarify the Eclipse Modeling Framework and its meta-model.

1. **Installation** and **Workspace** map to the class definitions of Java and UML and to the complex type definition of XML.
2. **location** and **lastWorkspace** are UML attributes, which need getter and setter methods in Java.
3. **workspaces** is a little more complex, as it is an UML association, which needs to be represented as a nested element declaration in XML, with another complex type. It might also need a getter method in Java.

This leads to the approximate Ecore model shown in Figure 2.4. An Ecore model is an instantiation of the Ecore meta-model. This example starts with the model, but EMF enables developers to begin with whatever they like, Code, XML or the model. The given model could now be serialized into a XML Schema, which is realized as Ecore XMI file in EMF. Before EMF can generate Java Code from the model some parts need annotations. This is especially true if the component or interface are not well defined to be used as a model definition. The `{@model` annotation in the

JavaDoc explicitly tells EMF that this method is part of the model definition. This is shown for the *Installation* interface in Listing 2.1.
It is important to note, that EMF generates not only interfaces, but also the implementation and test classes. The generated interfaces extend the `EObject` interface which is the `java.lang.Object` pendant for EMF. This interface allows to get an object's metaobject or the container containing the object and it extends the Notifier interface itself. EMF supports model change notification for EMF objects, if other objects that are observers require a notification [EMF09].

```
/**
* @model
*/
public interface Installation {
  /**
  * @model
  */
  String getLocation();
  /**
  * @model
  */
  String getLastWorkSpace();
  /**
  * @model type="Workspace" containment="true"
  */
  List getWorkspaces();
}
```

Listing 2.1: Installation Interface example with Java annoations for EMF. The annotation explicitely include the methods in the model.

The significance of EMF for the Eclipse Installation Manager will be explained in the development chapter. In short, the application uses and already existing EMF Model and utilizes the capabilities of the framework to identify resource types and check if they are consistent with their definition. Additionally, the modification of the resources present on the system is done through the EMF which ensures consistency of the resources states with the model.

2.1.5 Eclipse Oomph Project

The Eclipse Oomph project is a collection of tools that provider the user with various possibilities to adjust Eclipse Platform preferences, describe IDE configurations, manage bundle pools and more. Especially interesting are EMF models, which the projects provide. These EMF models can be loaded by code and, together with the Eclipse Modeling Framework, this enables the developer to verify existing model resources against the model and recognize their type reliably. In the following section the Eclipse Installer will be explained followed by a connection between the installer tool and the EMF model [EOP].

Eclipse Installer

The Eclipse Installer is on of the tools provided by the Eclipse Oomph project. It utilizes Setup models which are represented as `.setup` files and contain the product or project models. The product model consists of **setup tasks** which refer to **scopes**. There are multiple local scopes defined below. Scopes can also be nested, in which case they inherit tasks of parent scopes.

Installation scope References one product version scope, which applies all tasks apply to the specific installation and also contains all tasks for the installation.

Workspace scope Mostly contains workspace related tasks, but can point to Stream scopes, which apply Stream tasks to the workspace

User scope Contains user specific tasks applied across all installations and workspaces.

There are more types of scopes, but the most important for this project are the product catalog, product, project catalog and project scopes, which are defined below.

Product catalog Contains product scopes either directly in the file or as linked file.

Product Contains product specific tasks and Product Version scopes.

Project catalog Analogous to the product catalog, the project catalog contains project scopes and tasks directly or as linked files.

Project The project scope contains Streams and some tasks for the scope.

While this clarifies what a product is composed of, it is not necessarily more understandable. Practically a product file is a configuration of an Eclipse application, composed in a file using the XML Schema. Tasks in an Oomph product usually refer to setup tasks, which can include requirements and their dependencies as well as simple configuration of the application. Examples for product setup files and a product catalog can be found in the BootPalladio repository[1]. The Setup model is customizable and supports extended features like Variables, Conditional Tasks or Macros, and it is at the core of the Eclipse Installer. Any Eclipse Application can be delivered as product file [EOA]. A project represents a project configuration and, for example, enables the Eclipse IDE to directly clone and import a project into the IDE from a repository.
The Eclipse Installation Manager targets installations made by the Eclipse Installer, which base on these Product or Project Setup files. Understanding how the Eclipse Installer works is essential. The EIM might be implemented as an Eclipse Plug-In, used by the Eclipse Installer. If accepted, the Eclipse Installation Manager can be a contribution to the Eclipse Oomph Project increasing the installers capabilities.

2.1.6 bnd/bndtools

bnd is a tool that uses other popular build tools like Gradle[Gra] or Maven[Mav] to resolve dependencies, build the application and generate the metadata necessary. The basis of this is a workspace model and a project model. **bndtools** is the implementation of bnd as an Eclipse Plugin-In. bndtools enables automatic builds from inside the Eclipse IDE.

[1] https://github.com/A7exSchin/bootPalladio

A bndtools workspace is characterized by a `cnf` folder, which includes a cache, bundle repository folders, external configurations and a central configuration file `build.bnd`. This file configures the repositories, bundle versions and various other things, regarding the bundle and its build process. A bndtools project on the other hand is characterized by its `bnd.bnd` file and can be run with a `name.bndrun` file. The first file configures which packages are private, which can be exported (which means, accessed from outside the bundle) and which imports are available in the java classes. The imports are managed with a **bnd Bundle Path** and the bundles are located in various caches, like the local .m2 maven cache or the cache inside the workspace.
bndtools differentiates between build dependencies and run requirements, where the latter are only necessary during runtime and the build dependencies are only necessary during build time. The configuration also allows specific bundles to be blacklisted from being imported. The run requirements can be resolved from inside `*.bndrun` files, allowing for different run configurations. For example a developer can create a `headless.bndrun` file and a `app.ui_win32.bndrun`. The first file runs the app in a headless state with a shell bundle as a consumer, the second one also includes a UI bundle which serves as consumer. Any `*.bndrun` file can be exported with Gradle, as bndtools comes with necessary build tasks. Exporting results in an executable java archive.
The last important configuration possibility is the OSGi framework. Currently there are two commonly used frameworks, `org.eclipse.osgi` and `org.apache.felix.framework`. Both are OSGi compliant and can be used interchangeably.
bnd with bndtools is used to simplify and streamline the development process and project building. Dependency management is made easy and the developer does not need to care about the explicit generation of metadata like with PDE. Additionally bndtools provides capabilities to release bundles to a release repository, including managing their version adhering to the OSGi Semantic Version specification.

2.2 Related Work

This section discusses how research was done to find relevant software as well as the results of this literature search.

2.2.1 Literature Research Methodology

As this thesis proposes a new tool most of the literature research was done using online search tools, like Google and Duck-Duck-Go. Searching with terms like *"Eclipse manage installations"* lead to an older discussion[2] showing that the proposed question and problem is not a new one, but that no efficient solution has yet been proposed. Results on later pages were dismissed as their relevance diminishes. Additionally, Google Scholar[3] was used for literature research. One relevant paper was identified which examines criteria of Eclipse-based tools in a industrial environment [GRDL09]. However, this paper focused on implementation efforts and did not feature other software products.

[2] https://stackoverflow.com/questions/3799642/how-do-you-manage-your-eclipse-installation
[3] https://scholar.google.de

2.2.2 Yoxos Installation Manager

An application with a similar idea to the Eclipse Installation Manager is the *Yoxos Launcher*. It is a closed-source eclipse software that has been developed by EclipseSource GmbH. The software focused on custom Eclipse configurations and enabled to start, create and share Eclipse Configurations with other developers. It utilized P2 repositories to add own Plug-Ins into the Eclipse Installation, but the latest version found is 5.6, which was released in March 2013 [YX13]. At the time of writing Yoxos was not present at the products section on the website of the company. EclipseSource GmbH was contacted to ask about further development of the software and a test sample to be able to compare their product to this proposal but they have yet to reply.

2.2.3 IBM Installation Manager

The IBM Installation Manager is a similar tool compared to the Eclipse Installation Manager. However, the focus of the IBM Installation Manager is the installing, updating and modifying installations. Additionally, the IBM Installation Manager can roll back installations to a different state. IBMs solution is based on repositories. A repository in this context is either one of `diskTag.inf` file, `JAR` file containing a repository, `repository.config` or a `.zip` file [IBM]. During the research no repository was available because they are only accessible if you have a valid IBM rational product which supports installation with the IBM Installation Manager. To do so the user selected the downloaded `.zip` file as repository which allowed the user to install the product.
In the end both tools try to help the developer with managing their installations. However, the IBM Installation Manager only works for certain IBM products. In addition it is closed-source and IBM provides no documentation to create your own repositories. IBMs tool provides more functionality, like updating your installations, compared to the Eclipse Installation Manager, but lacks other features like a comprehensive overview over the installed installations. Both concepts could potentially benefit from each others ideas and implementations.

3 Requirements Engineering

This chapter addresses the process of collecting and processing of requirements as desired by users of the tooling. Formally, Requirements Engineering is a structured approach to capture, analyze and manage requirements a piece of software has. This process starts by recognizing the presence of *stakeholders*, who are individuals with a vested interest in the software. Especially important is the communication between stakeholders to reach consensus about the requirements. A requirement can be defined as a property or capacity that a system or person needs to fulfill or have, in order to satisfy a contract, specification or other documents [Chr14]. A documented representation of a property is also considered a requirement [90]. Requirements Engineering can be divided up into four parts:

1) Requirements Elicitation
2) Requirement Documentation
3) Verification and Reaching consensus
4) Requirement Management

Each of these parts includes explanations on why they are important and what was being done in respect to the Eclipse Installation Manager project.

3.1 Requirements Elicitation

The requirements elicitation begins with the stakeholder identification. It is important to categorize different requirements from various sources and put them into the context. Stakeholders are defined by their importance and their influence on the development process of the product. The Eclipse Installation Manager has three stakeholder types: customers, users and the developer. The stakeholders can be displayed in a coordinate system according to their influence and importance to show differences between them. In Figure 3.1 each stakeholder has been evaluated by their actual influence that they had on the project's development process. For example users usually do not have any influence on the development of software, but they are very important as the tool aims to address their needs. It is important to note that the user group is potentially unlimited. Only a small group of users were involved in the development process which included representatives of the important user groups, e.g. Palladio developers, Java developers using the Eclipse IDE. The Table B.2 contains a list of all representatives of the stakeholder types that were queried during the development.
The customers are Prof. Steffen Becker from the University of Stuttgart and Mr. Peter Kirschner from the Kirschners GmbH. This type of stakeholder is of high importance but has a low influence, because they are no developers themselves. Additionally, both customers are users of the product,

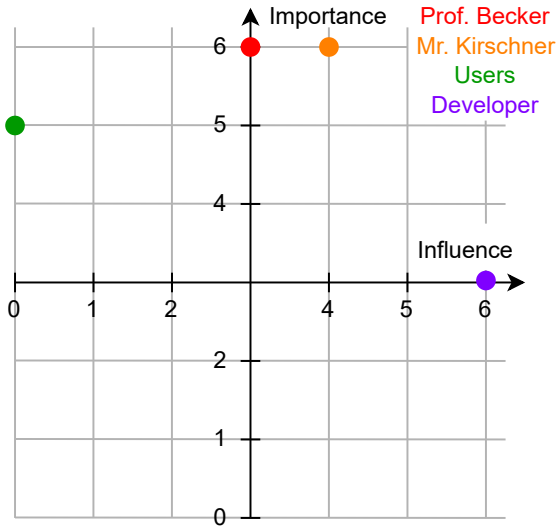


Figure 3.1: Categorization of the stakeholders for the Eclipse Installation Manager in terms of their influence on the development and importance for the project.

so their classification needs to account for that. In this case both have a moderately high influence on the development process, as Professor Becker needed the software to work with MacOS and Mr. Kirschner helped when technical questions arose during the development. From the user group of stakeholders two postdoctoral researchers were chosen based on their interest in the project.
Typically stakeholders do not provide well-formed requirements because while they are domain experts, they do not need to be development experts. For this reason there are suggested methods for the requirement elicitation, depending on various properties of the stakeholders, like availability, communication abilities or human interactions. For the customers the interview method was chosen, because that information gathering technique is applicable when the stakeholder type does not have a lot of time or only a set of roughly defined requirements is needed [Chr14]. As these interviews were the first discussion for this project a rough set of requirements was necessary to define the idea and scope of the thesis.
To represent the users perspective, additional requirements were gathered using the brainstorming method with two example users. The set of gathered requirements now needed categorization and prioritization which was done according to the Kano Model. This model classifies product features based on a Kano survey into five categories and is used in product development and user satisfaction analysis [Chr14].

1. Must-be Quality: This category contains features that are expected by the users.
2. Performance Quality: Features in this category influence the user satisfaction linearly; the better they are implemented, the higher the user satisfaction
3. Excitement Quality: Very well received features that excite the users, but they are not breaking if nonexistent
4. Indifferent Quality: Product features users do not really care about

5. Reverse Quality: This category includes product features that, if present, lower user satisfaction

Results from the Kano survey are mixed. On the one hand performance and excitement qualities are clear: displaying and searching installations and workspaces are the top features users like to see. Excitement arises from the product's capability to initiate and delete installations and workspaces, along with the anticipation of the product's inclusion in the official Eclipse Installer. On the other hand a lot of qualities that seem to be necessary were not perceived by the users to be so, namely adding manual Eclipse Product Installations or the configuration of installations and workspaces. The main reason for these mixed results could be that some questions were not properly read and answers were given wrong. For example one participant strongly disliked both functional and dysfunctional questions in the survey. The results of the survey are gathered in Table B.1. Each feature is listed together with its assigned category and the specific answers of the Kano survey participants. Table B.2 contains the stakeholders alongside their Role, Description, Availability and Area of Profession. The table also includes the reason why the representative of the user role was chosen. After the discussion of the quality and feature categorization according to the Kano model a user pointed out that, since there is no comparable software available, the expectations were low to begin with.

3.2 Requirements Documentation and Verification

With the basic ideas of qualities and features finished the next step is a proper documentation of requirements. With an agile approach for this project user stories are the preferred format of documentation. Additionally all qualities have been listed in the public discussion part[1] inside the project's Github repository. The goal was to provide all users a platform to propose new features and discuss current ones. A user story describes a feature that is relevant for the customer or user of the product. This includes a description for the feature and user stories need to be reassessed with the stakeholders, if they are complex [Chr14]. User stories also include a point of view, which usually is a role, a task description what the role wants to achieve, and a goal which describes the benefit for this role. Furthermore acceptance criteria confine the space of possible solutions and what needs to be fulfilled in order for the product to be accepted by the customer. These acceptance criteria follow the SMART formula: **S**pecific, **M**easureable, **A**chievable, **R**elevant, **T**ime-boxed [Chr14]. The user stories in Table 3.1 and Table 3.2 are examples for two of the main functions the Eclipse Installations Manager is supposed to fulfill. The first one describes an overview feature which the user wants and what the user expects when the overview is shown. The second use case is about the tools ability to start installations with specific workspaces. User stories can be implemented in different ways depending on the and the implementation for the first use case is shown in Figure 4.3 in a later chapter.
It is possible to increase the amount of details in the documented requirements when using the Given-When-Then schema or Use-cases and Story-maps. As the focus of this work is not the art of Requirements Engineering, but the development of a useful software and the available time is limited user-stories were chosen as a compromise when documenting the users needs and planned features. However, two general use cases have been identified but not documented formally:

[1] https://github.com/A7exSchin/EclipseInstallationManager/discussions/29

Table 3.1: User story about the ability to provide an overview over the installations present on the machine. This user story is one of the main benefits the Eclipse Installation Manager aims to provide and symbolizes a key feature.

As	User
I want to	be able to see all existing Eclipse IDE installations on my machine.
so that	I can better keep an overview over my development tools
Acceptance criteria	• A list of all available installations • Installations should be identifiable

Table 3.2: User story about the ability to start any installation and workspace. This provides developers with a quick access to their development environments.

As	User
I want to	be able to start any available installation with assigned workspaces
so that	I can use my tools faster and more consistently without the need to search in the file explorer
Acceptance criteria	• The installation view should include a button to launch said installation • The launch should work on multiple machines and operating systems

1) day-to-day use of the Tray Application. It is expected that most of the time developers use the EIM to see their installations and start them using the Tray Menu Overview. With this use case no management options are needed.

2) Management Use case. While not used often, the management aspect of the EIM is also providing important functionalities.

The finished product needs to pay attention to both use cases, as they put different requirements on the design of the user interface.
The last important part is the verification and validation of the requirements. In weekly meetings the development and progress regarding requirements was verified and the implementation was validated, effectively making sure the correct product was built.

3.3 Requirements Management

Requirements Management is the process of collecting and displaying the gathered requirements in an understandable and precise way. One of the main focuses of this part is that every person interested in the project should be able to work with these requirements. As the Eclipse Installation Manager is a small Bachelors Thesis project done by a single developer, Requirements Management has been limited.

4 Development

This chapter discusses all implementation and development efforts as well as their limitations. This includes the architecture design, the code itself, development operations and any other additional work necessary. As the main goal of the thesis is the development of the Eclipse Installation Manager the most time was afforded to this task. Special efforts were made to make the documentation understandable and the tools itself user friendly.

4.0.1 Palladio Integration

One of the biggest requirements to the Eclipse Installation Manager was the capability to work with the Palladio Bench tool. This goal was reached by creating an additional repository[1] which contains a Oomph Product Setup for the Palladio Bench and a product catalog, which can be consumed by the Eclipse Installer. To create such a Product Setup the Eclipse Features which define the product are needed. While these can be found at the Palladio Updatesite[2] the overarching Palladio Product feature was missing. It is to be noted, that only installing the functional features results in a functioning Palladio instance, but lacks any customization done with the product, like configuration and branding. With the help of the Palladio development team the product feature was added to the Palladio site for the current release. Ultimately, developers can now install the Palladio Bench from the Eclipse Installer, after adding the product catalog from the `bootPalladio` repository to their local Eclipse Installer installation.

4.1 Architecture

The Eclipse Installation Manager consists of multiple bundles, which can be loaded into an OSGi framework. The most important bundles are

1) An API bundle, that defines the Eclipse Installation Manager service interface and data structure.
2) An implementation bundle, which implements the API and adds relevant functions.
3) An application bundle, which utilizes the implementation bundle and adds a user interface in the form of a system tray icon and multiple view components. It also contains the data provider for the user interface.

[1] https://github.com/A7exSchin/bootPalladio
[2] https://updatesite.palladio-simulator.com/palladio-build-updatesite/releases/

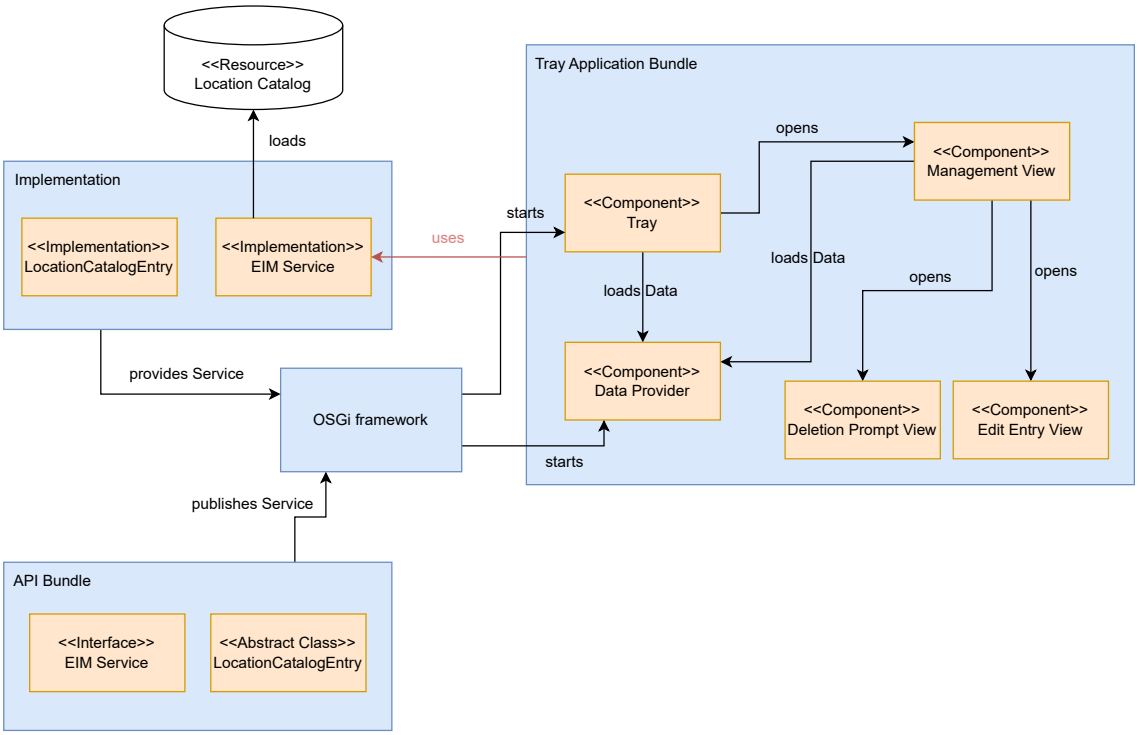


Figure 4.1: The component architecture of the Eclipse Installation Manager including the local location catalog resource that is used to fetch the necessary data.

This structure can be seen in Figure 4.1 and was chosen because of all the benefits already mentioned in the foundations chapter which the OSGi platform provides. This also simplifies a possible adoption by the Eclipse Oomph project. However, the structure comes with implementation drawbacks. One of the most time consuming implementation hurdles was the understanding how OSGi components work within the specification rules. Due to the huge amount of capabilities of the OSGi specification it can get a bit complicated. Although some prior experience was present, the correct utilization of the component runtime and service declaration to only activate certain user interface elements if they are needed took some time to understand. In the end benefits like modularization outweighed the drawbacks.
As already mentioned, the Eclipse Installation Manager uses resources that were generated by the Eclipse Installer, namely its location catalog. The location catalog is a XML file which consists of a list of installations. These installations are mapped to their respective workspaces. For each installation or workspace, only a URI in the form of a file path was given. For that reason the data needed to be mapped to a Java class which is called `LocationCatalogEntry` and is contained in the API bundle. It represents one pair of an installation and a workspace. This design was chosen because the user interface required views for installations and workspaces separately, while not loosing their mapped information. One instance of a `LocationCatalogEntry` can represent a workspace or an installation.
The reference application bundle, which contains the user interface, is implemented using the Model-View-Controller (MVC) approach:

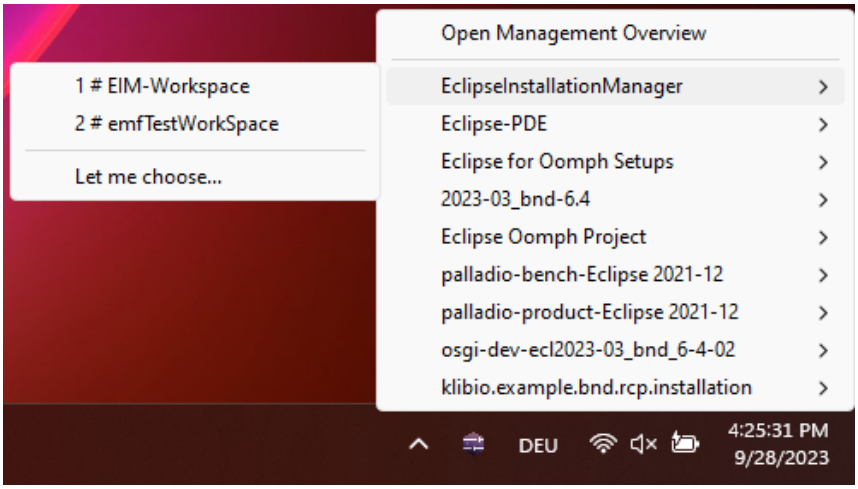


Figure 4.2: Screenshot of the system tray components main menu. This can be opened by left clicking the system tray icon. Each installation is shown with the respective assigned workspaces and can be launched by clicking on said workspace. The Management View can also be opened from this menu.

- **Model:** A data provider component acts as model and loads the list of location catalog entries using the reference implementation and prepares the data so the user interface can easily display it.
- **View:** A component which creates the system tray application uses multiple other view components to display the data loaded from the data provider.
- **Controller:** External program logic needed is provided by the implementation bundle and accessed from the View via the Declarative Services capabilities.

This structure fits very well with the modularization the OSGi framework provides: Each of the MVC components can be directly implemented as an OSGi component which provide their service to the other components. It is also important to mention that the application logic is contained in the implementation bundle, not inside the tray applications bundle.
The user interface consists of multiple components. The first one is the system tray component, which is the main entrypoint for the user and provides an overview over installations in a list. It is shown in Figure 4.2. Apart from this main menu the tray component also features a secondary menu for settings and closing the application. Each installation entry in the list can be expanded to see assigned workspaces. Clicking on a workspace opens the installation with this workspace set and alternatively you can choose your own workspace. This component accommodates the aforementioned daily use case and provides the developer with a small and quickly usable overview over the existing Eclipse installations. For the second use case, where a developer needs to delete and rename said installations the Management View component was created. It contains a small view with a search bar at the top and an installation as well as a workspace tab. This view is shown in Figure 4.3. Each installation or workspace can be renamed or deleted. For both of those actions small additional user interface components were created which can be seen in Figure 4.5 and Figure 4.4 respectively.

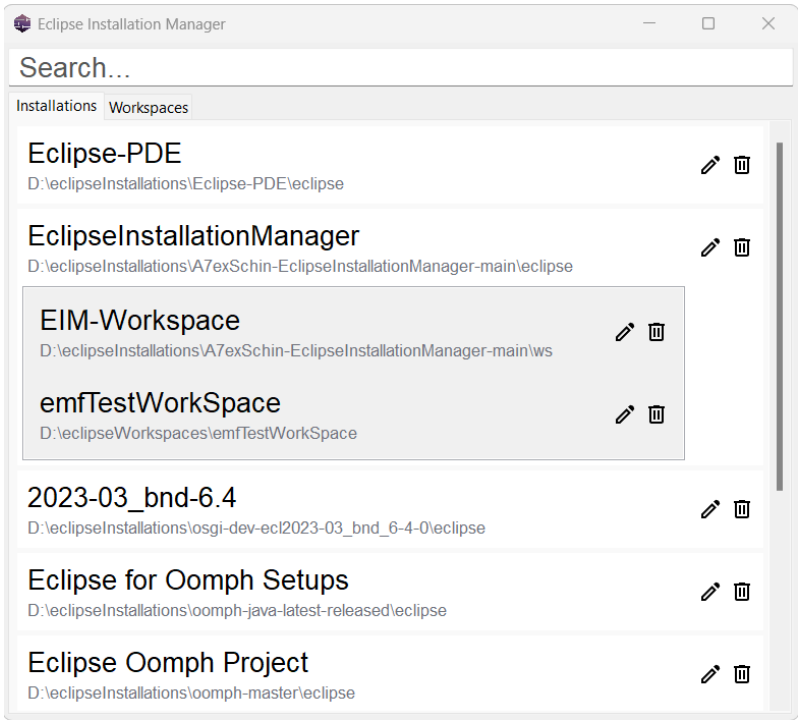


Figure 4.3: The Management View provides the user a searchable list to quickly find, rename and delete installations and workspaces. This window can also be used to start installations by left clicking a workspace entry.

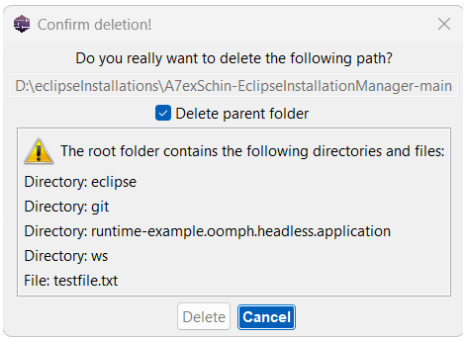


Figure 4.4: Prompt to confirm the deletion of an installation folder. In this case the user toggled the `Delete parent folder` option and gets a warning which displays the contents the user is about to delete. The `Cancel`-Button is highlighted and selected by default to prevent users to accidentally delete installations by just pressing enter.

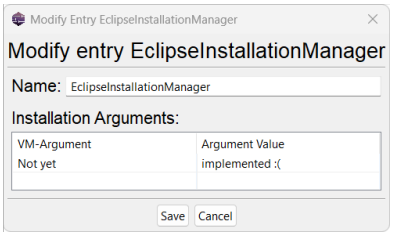


Figure 4.5: Small dialog window to offer the user the capability to change the name by which the Eclipse Installation Manager displays a certain installation or workspace. This window could potentially support per installation environmental variables.

The user interface was designed to be quickly navigable and put emphasize on ease of use.

Leveraging the Setup model

The Eclipse Modeling Framework can map existing resources to Java classes defined by an EMF model. As an example take a look at the `locations.setup` file, which the Eclipse Installer creates in the `.eclipse` folder in the systems user home. Figure 4.6 shows a fraction of the Oomph Setup model

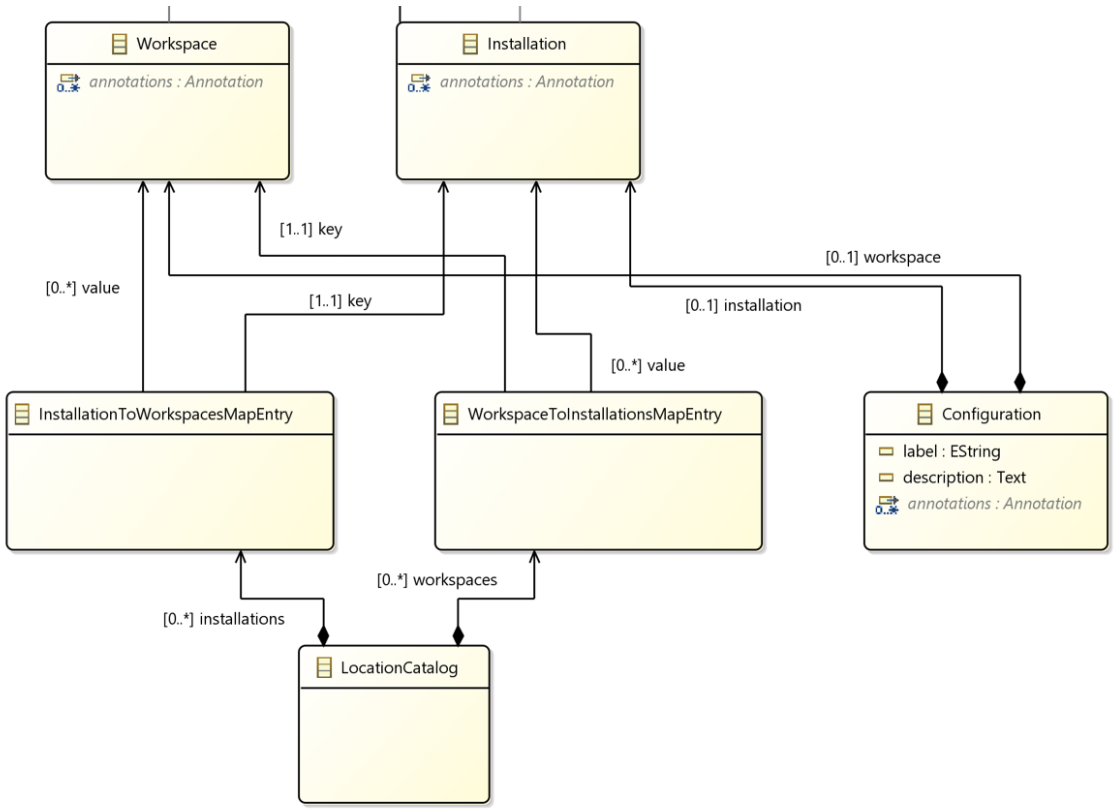


Figure 4.6: Small part of the Eclipse Oomph Setup Model which shows the most important parts for the Eclipse Installation Manager.

with the most important classes for the Eclipse Installation Manager. As shown the model contains various classes including `Installation`, `Workspace`, `LocationCatalog` and `Configuration`. EMF can now check if a resource matches the models class definition given the model and the implementation. The location catalog contains `InstallationToWorkspaceMapEntries` and `WorkspaceToInstallationMapEntries` which can be iterated through. The installations and workspaces are defined with an Universal Resource Identifier (URI). Those URIs point to the respective Setup files for an installation or workspace. This is the basic workflow of the Eclipse Installation Manager: first, load the location catalog file as a resource and check if it matches the models definition of a `LocationCatalog`. With the URI contained in the map entries the Eclipse Installation Manager can start any Eclipse Application installed with the Eclipse Installer.

Additionally, the application is up-to-date because the Eclipse Oomph Tooling includes an engine that keeps the location catalog updated and, for example, adds a new entry, when the Eclipse IDE has been started with another manual workspace.

4.2 Documentation and development operations

The Eclipse Installation Manager was created to be used publicly. For this reason additional work was put into a flawless build and publishing process as well as a small website to improve the presentation.
To build and publish the build artifacts a Github workflow was set up which was split into those two parts. The build script executes tests and if they run successful will continue to build the application. Most of the releases contain executable Java Archives (`*.jar` files) which is planed to change in the later releases to platform specific executables, e.g. `*.exe` files for windows and `*.app` files for MacOS. This requires additional infrastructure to sign and upload created bundles to Maven Central, an extensive software repository. Additionally based on the released bundles a Maven Tycho build is executed to generate those platform specific packages. All of that was done to improve the ease of installation and use for the user.
After a successful build the release script creates a Github Release and adds the generated build artifacts. Releasing the API package to Maven Central allows other developers to use this API and create their own implementation. Each release has a tag which corresponds to the version of the tray application and versions of all bundles follow the principles of Semantic Versioning.
The website[3] uses Github pages and a Jekyll theme to graphically display relevant information like installation and usage instructions. While not strictly necessary, the time expenditure was minimal and therefore deemed worthwhile.

4.3 Missing Features

Several features, that came up during the Requirements Elicitation could not be implemented. The main reason was the time constraints put on the project as well as the prioritization of more valued features. Other features had additional implementation obstacles which could not be overcome or just did not make a lot of sense in the context of the Eclipse Installation Manager. A prime example for this would be the feature to configure an installations `eclipse.ini` file. This was not implemented because installations are based on Oomph Product Setup files. These setup files should include any setting that is necessary for the product. Additionally the Oomph Setup Core will overwrite any settings changed by hand, to keep the state of the installation consistent with its Setup file. It is therefore recommended to change any settings inside the Setup files. Similarly some features tried to solve a task that was hard to implement but easy to do themselves, like the addition of own Eclipse Installations. As already mentioned, the Eclipse Installation Manager uses a location catalog file created by the Eclipse Installer. This file contains every installation and is maintained by the Oomph Setup Core at every start of an Eclipse Application Instance. Adding a new installation is as easy as installing the Oomph Setup Core plug-in into the Eclipse instance. After this it will be

[3] `eim.a7exschin.dev`

automatically added to the catalog on restart. Therefore the implementation has been skipped and a How-To guide has been created[4]. All these features together with the reason, why they were not implemented, can be found in Table B.3.

4.4 Improvements

Given enough time the Eclipse Installation Manager could be vastly improved. There are multiple capabilities which can be implemented to provider better compatibility with industrial environments. The user interface can be improved upon too, as it currently is programmed using the low level Java Standard Widget Toolkit and uses a lot of interface resources on Windows machines. The usage of JavaFX could alleviate this problem and improve design and usability of the user interface. Furthermore the Eclipse Installation Manager in its current form is only well tested on the Windows operating system. While the MacOS and unix versions are functioning, they come with their own issues. For example on an Apple Macbook with aarch64 architecture and MacOS there are multiple issues regarding the dark mode and delayed UI responses when refreshing the data.
Other improvements would be possible by adding more features. The biggest one would be the integration of the Eclipse Installation Manager into the Eclipse Installer. For now, the current code base of the Eclipse Installation Manager is not mature enough to be integrated in the official bundle. To still provide an integration, the option to quickly launch the Eclipse Installer was added. However, this feature came with problems itself, because MacOS applications usually do not use double click events in the system tray. Therefore a separate way of opening the Eclipse Installer on MacOS and Linux was introduced: `Option Key + Left Click` or `Alt Key + Left Click` respectively.
Another feature which would improve the Eclipse Installation Manager is an integration of the P2 director capabilities. These implementations are more in-depth and required more time than what was available, but would enable the tool too install Eclipse plug-ins from P2 repositories into existing Eclipse Installations. Similar to that an implementation of the Oomph Setup Core could benefit the refreshing of the location catalog. Currently the location catalog is only updated every time an installation is started, so inconsistent states could arise after deleting a resource and before starting an installation instance. As last point of improvement is the unit and integration testing. Some unit tests are present, but these are by no means extensive or sufficient.

[4] https://github.com/klibio/EclipseInstallationManager/discussions/35

5 Evaluation

To answer the research question, if the Eclipse Installation Manager, can help developers by reducing the time and efforts needed to manage their environments, an evaluation is necessary. This evaluation will gather the relevant data from a small study to either verify that the Eclipse Installation Manager improves developer productivity or refute this claim. This section will go into detail about the design of the study, the results and a discussion about the validity of those results.

5.1 Goal-Question-Metric Approach

The Goal-Question-Metric (GQM) approach helps with identifying valuable questions and metrics based on the given goal [BCR94]. As already mentioned, the goal of this thesis is to examine if a tool like the Eclipse Installation Manager can influence the process of Eclipse developers. As GQM defines goals based on the type of objects, multiple goals need to be defined to accommodate the scope of the impact of this project. The first goal focuses on the changes the product has on the development process. The second goal refers to the product itself: the Eclipse Installation Manager and the questions focus on the deliverables and documentation. Therefore our main goal can be split up into these two goals:

G1 Create a concept to improve the workflow of developers.

G2 Create a software product that enables developers to manage Eclipse Installations.

Now questions can be formulated that need to identify the achievement of these goals. It is important that questions refer to a selected issue. Based on the given goals, the following questions were chosen:

Q1.1 What problems make the current development process undesirable or inefficient?

Q1.2 Which of these problems are alleviated by the Eclipse Installation Manager?

Q2.1 Does the product fulfill the desired function of helping developers in their daily workflow?

Q2.2 Is the user interface understandable and easy to navigate?

Q2.3 Is the documentation sufficient?

The questions **Q1.1** and **Q1.2** address the process issues and solutions of the current development process, and how effective the Eclipse Installation Manager deals with those problems. The questions **Q2.1** to **Q2.3** address the product itself, especially how well it is designed and how usable it is. Both of those categories affect the results and the efficacy of the project.
The development process is different for each user, so the data that was collected during the survey is mostly of a subjective nature. Further evaluations in this topic might go deeper with more objective

metrics, for example if the time to execute specific tasks, like deletion and starting, was measured and compared. As the main focus of this thesis is the development of the tool only a small survey was done to provide data to answer the research questions.

5.2 Study Design

To consult users on the efficacy of the Eclipse Installation Manager multiple guided peer reviews were done. Those peer reviews included the tools developer and a user with experience using Eclipse Products. These users were chosen because the Eclipse Installation Manager currently manages only installations done with the official Eclipse Installer. Additionally the study includes users which were familiar with the tool as well as completely new users. Multiple ways were used to reach out to possible participants, including creating a post on the Eclipse Community Forums, a post on the Eclipse Reddit Community, a Github discussion on an Eclipse Project Github Repository and a post using the Telegram messenger in the Telegram group for Data Science, Informatics, Media Informatics and Software Engineering students of the university of Stuttgart. Additionally, a mail was written to the Palladio Developer mail list and people at different institutes from the university of Stuttgart were asked. While the overall reach of the posts was significant (1100+ views on the Eclipse Forum) the engagement was low and only 9 people agreed in participating. A list of tasks that needed to be completed with the Eclipse Installation Manager by the user was prepared beforehand and the user was provided with the public installation and usage documentation. During these tasks the user was allowed to ask question regarding the usage and capabilities of the tool. It is important to note that some users did not use the official Eclipse Installer before this thesis. Hence participants needed to install the installer before the study began.

The task list for the Eclipse Installation Manager is

1) Install the Eclipse Installation Manager from `eim.a7exschin.dev`.
2) Set up the Eclipse Installer Location.
3) Install a Palladio or an Eclipse Instance.
4) Rename an installations.
5) Start an installation but choose a new workspace.
6) Rename a workspace.
7) Delete a workspace.
8) Search for an installation and delete it.

After the tasks were fulfilled by the participant they were given some time to ask questions. With no questions left a questionnaire needed to be filled out to gather the necessary measurements. Subjective questions, e.g. *"How well does the Eclipse Installation Manager address the problem of getting a proper overview over Eclipse installations?"* needed to be rated on a 5 point Likert-Scale from *"Not at all"* (1) to *"Very well"* (5). In addition, open text questions like *"How do you think the Eclipse Installation Manager will influence your development process?"* were asked to give the opportunity for more detailed feedback on specific topics. These questions are based on the goals and questions of the GQM approach and can be found in Table B.4. Ultimately, the

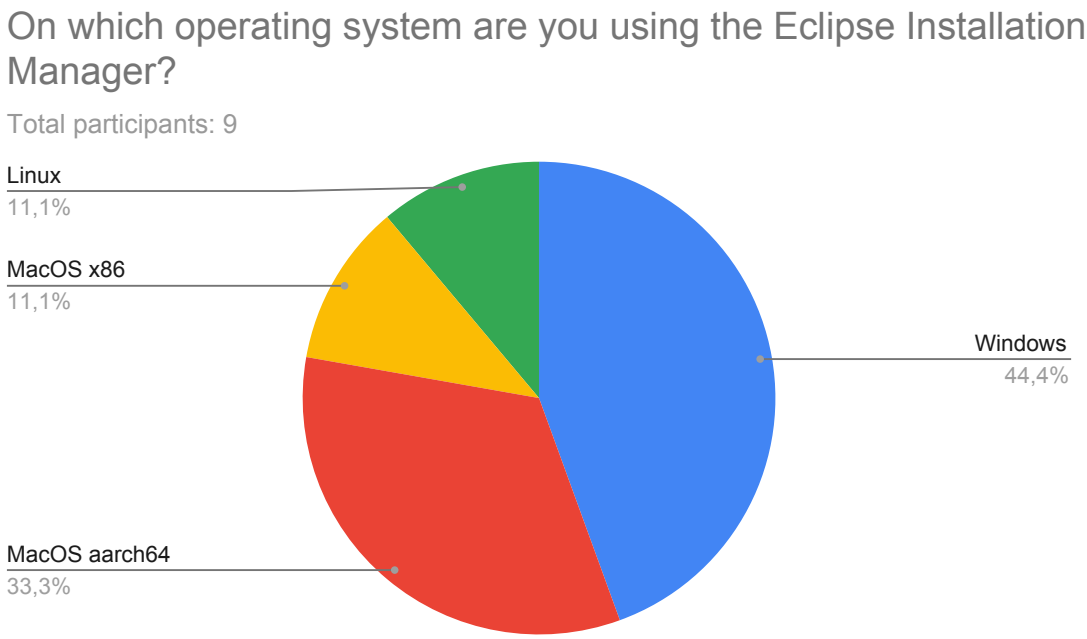


Figure 5.1: Operating systems used by participants while they were using the Eclipse Installation Manager for the study.

questionnaire aims to evaluate the impact on the developers workflow as well as how efficient the Eclipse Installation Manager solves the problem. Additionally, most subjective questions target how the developer feels about using the tool. The next section presents the results of the questionnaire in an understandable way.

5.3 Results

This chapter presents the results of the survey and is followed by a discussion. There were a total of 9 participants and for each question which used a Likert-scale the mean value was calculated. Questions which needed a text answer were summarized. All users understood the general idea of the Eclipse Installation Manager. Some users even showed excitement to see their problem solved. Out of these 9 participants 4 were using the Eclipse Installation Manager with the Windows operating system, 4 with MacOS and one with Linux, as shown in Figure 5.1. All participants have had prior experiences with the Eclipse Installer. Furthermore, all participants recognized the problem of keeping an overview of installations and workspaces and how the Eclipse Installation Manager helps with that problem. The average score on the effectivity of the Eclipse Installation Manager is 4.33 on a scale from 1 to 5 and participants rated the management capabilities of the Eclipse Installation Manager the same as the effectivity, 4.33. Every participant except one plans to continue using the Eclipse Installation Manager, however the impact on the workflow was appraised with 3.44 on average.

Table 5.1: The average ratings from questions that used the Likert-scale, from 1 to 5.

Number	Topic	Average rating
1	How much does the EIM help with getting an overview?	4.33
2	How well does the EIM solve the problem of managing installations and workspaces?	4.33
3	How intuitive is the user interface?	3.78
4	Rate the overall usability.	4.00
5	Rate the feedback on critical actions.	3.89
6	Rate the setup process based on the documentation	4.44
7	Rate the impact of the EIM on your workflow.	3.44

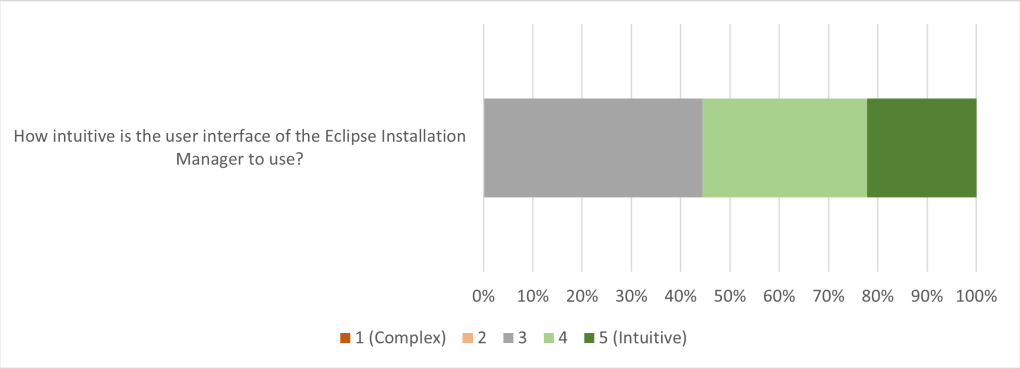


Figure 5.2: The graph shows percentages of how many of the participants gave the intuitiveness of the user interface a specific score. For example out of 9 participants two participants gave the UI a score of 5, three a score of 4 and the majority of four participants gave the UI a score of 3.

The general usability of the tool scored a 4.00 and the feedback on critical actions, like deleting an installation or workspace, was rated 3.89. The intuitiveness of the user interface was graded slightly lower with 3.78. The setup process which was provided by the documentation was assessed with 4.44. These results have been summarized in Table 5.1 and specific answer graphs can be found in the appendix, Chapter D. An example of a graph which represents the amount of different scores is found in Figure 5.2.

When asked about any options or settings a participant would have liked to see, three participants answered with an automatic refresh feature, while other answers referred to different features like better installation recognition using package managers, setting environmental variables for installations and providing functionalities to compare installations to one another. Additional remarks from the participants included either more feature requests, like support for global system hotkeys, language detection or an improved search based on additional metadata, or user interface improvement proposals, like adding an indicator that installations can be expanded to show their workspaces.

5.4 Discussion

Based on the Goal-Question-Metric approach this discussion considers the acceptance or rejection of the research question. Additionally, two hypothesis **H1** and **H2** were formulated based on this research question.

H1 The Eclipse Installation Manager helps developers managing installations and workspaces.

H2 The Eclipse Installation Manager impacts the workflow of developers in a positive way.

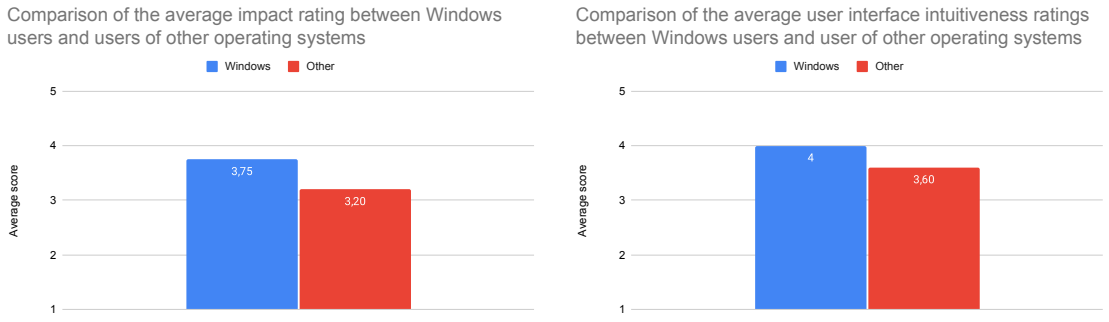
The questions formulated using the Goal-Question-Metric approach are used to discuss the concept and implementation. The answers to these questions lead to the goals given by that approach and ultimately evaluate the effectivity of the Eclipse Installation Manager.

5.4.1 Concept Discussion

To answer question **Q1.1** participants were asked which problems existed and they recognized multiple problems with their current development process. One of the most common problems in the development process is switching contexts between projects and keeping an overview of installations and workspaces. The overview problem is exaggerated by older installations and workspaces, which are either not needed anymore and clutter the local hard drive, or are still needed but hard to find. In addition, even if the installations are organized well, users either mixed up installations associated to a specific use case, or started the wrong workspaces as they usually were not named or identifiable.
Question **Q1.2** seeks information about the problems that the Eclipse Installation Manager can assist with. The application helped users to keep their local hard drive clean and one user stated that the Eclipse Installation Manager helps by reducing the amount of terminals that needed to be open, leading to a more clean desktop. Hence, all users appreciated the overview provided by the EIM, as seen by a relatively high average rating of 4.33. More over, the overview and renaming of installations and workspaces helped with separation of the different projects, effectively preventing confusion of responsibilities. The same holds true for the management capabilities the tool provides: Users generally liked the functionality, but had additional critiques regarding the execution. However, the users rated the impact on their workflow significantly lower. How this relates to the implementation is discussed in the next section.

5.4.2 Implementation Discussion

While users liked the application overall, the implementation of the concept was rated lower than the concepts itself. Intuitiveness of the user interface was rated 3.78 on average, which shows that there are improvements to be made, like adding arrow indicator to UI elements that are expandable. Furthermore, there is a bigger difference in the rating of the Eclipse Installation Manager depending on the operating system with which it was tested. While users of Windows rated the intuitiveness on average with 4.00, users of other operating systems like Linux and MacOS only rated it with 3.60. Similar rating differences occurred when asked to rate the impact of the Eclipse Installation Manager on the workflow: Windows users gave a higher average score with 3.75, whereas Mac users ranked it lower with a score of 3.20. These differences are shown in Figure 5.3a and Figure 5.3b



(a) Score for the impact on the workflow

(b) Score for the intuitiveness of the user interface

Figure 5.3: Comparison of the average scores given by study participants between participants who used the Windows operating system and participants who used other operating systems, e.g. MacOS or Linux.

and reveal a weakness in the implementation: Windows was the main operating system with which the application was developed. This especially impacts the user interface, as design decisions were made without enough experience handling different operating systems. Consequently, the other versions needed more bugfixing and workarounds which made the user experience worse.
Additionally, users often noted the lack of an automatic refresh of the user interface elements, when changes to the setup occurred. The current state of the application synchronizes the systems state via a location catalog file which is located inside the user home directory. This structure relies on a specific plug-in of the Eclipse Oomph project to update the file. For this reason the refresh functionality feels underdeveloped to the user. Ultimately, the answer to question **Q2.1**, whether the EIM fulfills its desired function to help developers, is yes. Nonetheless, users also noted that the implementation needs further polishing to improve the user experience. This applies not just to the functionality implementations but also to the user interface. The questions **Q2.2** and **Q2.3** about the user interface and documentation respectively, are directly answered by the survey results.

5.4.3 Research question

Based on these results both hypotheses, **H1** and **H2** can be accepted. The concept as well as the implementation work as intended. The research questions can be answered: The Eclipse Installation Manager helps users to get an overview of their installations and workspaces, and has a positive impact on the interaction between a developer and his/her development environment. Users tend to continue using the tool which acknowledges its value. Both goals, **G1** and **G2**, have been reached. However, the results also show the Eclipse Installation Manager needs more work before more developers will adopt the tool into their daily workflow. The current state of the software is usable, if the user is familiar with the existing problems.

5.5 Threats to Validity

It is common for experimentation in the software engineering field to classify the validity into four fields: Construct validity, internal validity, external validity and reliability [RH09].
Construct validity reflects whether the measurements that were taken during the study were understood correctly by all participants. To counteract problems regarding this aspect, users were allowed to ask questions during the questionnaire, if any questions were unclear. Additionally, the questionnaire and aim of the questions was clarified to participants before starting the survey.
Internal validity focuses on unknown factors of causal relations which might impact the results. In the first version of the study the question about the operating system was not present, but with the first test runs it became obvious that the operating system impacts the quality of the user interface as a third factor. For this reason a question was added to be able to visualize the differences for users of different operating systems. More over, a factor which was not recorded properly is the variance in experience of the participants with the Eclipse environment. This poses a threat to the validity to the claims regarding the impact of the Eclipse Installation Manager on a developers workflow. The participants were sourced from very different domains: university students, Eclipse project committers, Palladio developers and the broader Eclipse community were involved. All of these domains bring different amounts of experience and the tool impacts people who use the Eclipse IDE or Palladio Bench daily way more than it impacts people who just use Eclipse applications sometimes. This threat could be reassessed by limiting and separating participants and their evaluation of the tool and collecting additional data points regarding their experience.
External validity concerns the generalization of the results on a broader audience. For this study, the main problem here is the limited number of participants. Despite best efforts only 9 participants could be gathered. While this number is enough to get some feedback and general conclusions, it is not enough to support more specific claims. In addition, only one participant used the Linux operating system, limiting the significance of the study for all users of this operating system.
Reliability addresses bias which stem from the researchers. As the researcher for this thesis is also the developer of the Eclipse Installation Manager there naturally is a conflict of interest. As a developer one would want good marks and positive feedback relating the project. The biggest problem here is the way the introduction into the tool was given by the developer. While some procedure was defined, no two conversations had the exact same sequence. This can be addressed in future work with more specific metrics which are to be measured. However, the goal of this thesis is the implementation and the study serves as a small evaluation and verification of the concept.

6 Conclusion

6.1 Summary

This thesis provides a concept and an exemplary implementation to improve the workflow of Eclipse application users and developers. The application consists of multiple OSGi bundles and uses metadata the official Eclipse Installer creates to locate installations and workspaces of Eclipse applications. With this data the Eclipse Installation Manager gives an overview over all installations and workspaces in separate lists. As users need to be able to identify specific installations, the application additionally provides features to rename these resources. On top of that users can delete installations and workspaces using the tool. Furthermore, during this thesis necessary files which enable the Palladio Bench to be installed using the Eclipse Installer were provided in a separate repository.
The development of the tool began with requirements engineering, where requirements were gathered from stakeholders of the project. Based on those requirements the software development was started. Two use cases were identified and the user interface was designed accordingly. However, not all of these requirements could be implemented due to a lack of development time. A study was done to examine if the developed tool is able to positively affect the workflow of targeted users. The study was designed as a guided peer study, where the developer of the Eclipse Installation Manager introduced participants into the tool and they needed to execute a list of tasks using the software. This study was followed by a questionnaire in which participants rated how well the Eclipse Installation Manager addresses the stated problem as well as the software itself.
The evaluation showed that the concept idea certainly is valuable to developers. Despite that, it also became apparent that the current implementation needs further polishing in order to improve the user experience, particularly with regard to how the user's actions manifest within the application. In summary the contribution of this thesis to developers of Eclipse applications is the Eclipse Installation Manager. This application serves as reference for a novel solution to improve the daily workflow of developers relying on the Eclipse ecosystem.

6.2 Benefits

The benefits of this thesis mainly concern developers using the Eclipse IDE or those who develop Eclipse applications. Users of applications that can be installed with the Eclipse Installer can benefit by using the Eclipse Installation Manager, but currently not a lot of Eclipse applications support that setup process. However, the results of the study show that the benefits of the concept are substantial and only the implementation is lacking. Hence, given more time, the benefits of the application could be vastly improved. Additionally, this thesis shows that developers need and want tools that improve their workflow and might inspire additional work in that direction.

6.3 Limitations

Various different kinds of limitations apply to this thesis. Firstly, limitations to the software approach, where biggest limitation is the reliance on the Eclipse Oomph Project. A user looking to use the tool needs to have a certain setup including installations done with the Eclipse Installer. Any developer who does not want to use the installer, or install the Oomph Setup plug-in into their Eclipse instances cannot use the Eclipse Installation Manager. In addition the choice to use the Java Standard Widget Toolkit (SWT) puts limitations on the capabilities of the software depending on the platform it is running on. For example does the quick launch option not work flawlessly on MacOS and Linux and therefore a workaround was needed. Another limitation of this type would be a consistency problem, which appears when the installations or workspaces are deleted on the disk, because the location catalog file is only updated, when the Eclipse Oomph Setup Core plug-in is executed. This only happens on startup of an Eclipse Application which includes this plug-in. Secondly, the amount of participants regarding the study was limited due to time constraints. This restricts the value of the results.

6.4 Lessons Learned

Personally, with this project we wanted to apply the theoretical knowledge of the Software Engineering process. Doing this with an idea where the value was quite obvious on the beginning just made it more fun and we were eager to do it correctly. However, we have learned that Requirements Engineering is a whole science for itself. We can now understand how valuable proper requirement elicitation, documentation and management can be. Also communication is a key factor for any project, even if it is only taken care of by a single developer. Another important lesson would be that a developer should not have any fear to rewrite the implementation when it becomes apparent that the current code is insufficient or does not serve the purpose efficiently. Furthermore developing for various operating systems and system architectures poses additional problems, especially regarding the user interface. Different systems have different design ideologies which restrain the interface design and choices that are available to any developer.
Apart from those lessons, we also learned some technical lessons, like working with the OSGi framework or how to consume a large codebase and find what we need to use the parts we need.

6.5 Future Work

Current plans include the continued development of the Eclipse Installation Manager under the supervision of the Klibio GmbH. The company has an interest to provide this tool for companies and make it viable for restricted enterprise environments. However there is still a lot that can be improved and developed further:

- Improve the user interface to be more responsive and completely navigable without a mouse. Most importantly reevaluate the choice of the user interface framework used and reimplement it. Furthermore the implementation of the Java Native Library to work with the OSGi framework can help with a global keyboard shortcut to further improve usability.

- Hypothetically it should be possible to extend the Eclipse Oomph Model to include metadata for the Eclipse Installation Manager. For example an ID would enable the tool to recognize and save installation and workspace specific settings, like environmental variables at startup. For now, while these tools exist in separation, they cannot leverage each others whole capabilities without the need to circumvent different design decisions.
- Further development might include an implementation of the P2 director. This would enable the Eclipse Installation Manager to directly install the necessary Eclipse Oomph Setup Core plug-in into any Eclipse Application without the need of users to do it themselves. Similarly easily adding location catalogs to the Eclipse Installer is a feature which it is missing right now, but could be implemented so both features complement each other even more.
- From a scientific standpoint, this work only lays the foundation for more in depth research. More objective measures of the impact of various tools on the development process, like time until a developer is ready to do is daily work or amount of work done in lines of code could be examined.
- Inspired by the IBM Installation Manager future work could examine the possibilities to support updating, extending and rolling back existing installations.

Bibliography

[90] "IEEE Standard Glossary of Software Engineering Terminology". In: *IEEE Std 610.12-1990* (1990), pp. 1–84. DOI: 10.1109/IEEESTD.1990.101064 (cit. on p. 13).

[BCR94] V. R. Basili, G. Caldiera, H. D. Rombach. "The Goal Question Metric Approach". In: 1994. URL: https://api.semanticscholar.org/CorpusID:13884048 (cit. on p. 25).

[CEM] *Chapter 2. Presentation of the Ecore Metamodel*. kermeta.org. URL: http://www.kermeta.org/docs/org.kermeta.ecore.documentation/build/html.chunked/Ecore-MDK/ch02.html (cit. on p. 6).

[Chr14] S. Chris Rupp. *Requirements-Engineerung und -Management*. 6th ed. Hanser, 2014. ISBN: 978-3-446-44313-6 (cit. on pp. 13–15).

[EMF09] F. Budinsky, D. Steinberg, M. Paternostro, E. Merks. *EMF: Eclipse Modeling Framework*. 2nd Revised. The Eclipse Series. Addison-Wesley, 2009. ISBN: 0321331885; 9780321331885 (cit. on pp. 3, 6–8).

[EOA] *Eclipse Oomph Authoring*. Eclipse Foundation. URL: https://wiki.eclipse.org/Eclipse_Oomph_Authoring (cit. on p. 9).

[EOP] *Eclipse Oomph*. URL: https://projects.eclipse.org/projects/tools.oomph (cit. on p. 8).

[Gra] *Gradle Build Tool*. Gradle Inc. URL: https://gradle.org/ (cit. on p. 9).

[GRDL09] P. Grünbacher, R. Rabiser, D. Dhungana, M. Lehofer. "Model-Based Customization and Deployment of Eclipse-Based Tools: Industrial Experiences". In: *2009 IEEE/ACM International Conference on Automated Software Engineering*. 2009, pp. 247–256. DOI: 10.1109/ASE.2009.11 (cit. on p. 10).

[IBM] IBM. *Setting repository prefereces*. URL: https://www.ibm.com/docs/en/rational-clearcase/10.0?topic=manager-setting-repository-preferences (cit. on p. 11).

[Mav] *Apache Maven Project*. The Apache Software Foundation. URL: https://maven.apache.org/ (cit. on p. 9).

[OBL] *Announcement of Transition to Eclipse Foundation*. OSGi Blog. URL: https://blog.osgi.org/2020/10/announcement-of-transition-to-eclipse.html (cit. on p. 3).

[OSGi1] *OSGi Core Release 8*. OSGi Alliance. URL: https://docs.osgi.org/specification/osgi.core/8.0.0/index.html (cit. on p. 5).

[OSGi3] *Principles of OSGi*. aQute. URL: https://www.aqute.biz/appnotes/principles.html (cit. on p. 4).

[Palladio] *Palladio Software Architecture Simulator: Home*. Palladio Workbench. URL: https://www.palladio-simulator.com/home/ (cit. on p. 1).

[PK] P. Kirschner, D. Fauth. *Building Nano Services with OSGi*. URL: https://peterkir.github.io/ece2016/slides/Building_Nano_Services_with_OSGi_Declarative_Services.pdf (cit. on p. 5).

[RCP] *RCP Enable Eclipse to be used as rich client platform*. Eclipse Foundation. URL: https://bugs.eclipse.org/bugs/show_bug.cgi?id=36967 (cit. on p. 6).

[RH09] P. Runeson, M. Höst. "Guidelines for conducting and reporting case study research in software engineering". In: *Empir. Softw. Eng.* 14.2 (2009), pp. 131–164. DOI: 10.1007/s10664-008-9102-8. URL: https://doi.org/10.1007/s10664-008-9102-8 (cit. on p. 31).

[SemVer] *Semantic Versioning - Technical Whitepaper Rev. 1.1*. OSGi Alliance. URL: https://docs.osgi.org/whitepaper/semantic-versioning/Semantic-Versioning-20190110.pdf (cit. on p. 4).

[VM13] L. Vogel, M. Milinkovich. *Eclipse 4 RCP: The complete guide to Eclipse application development (Converted)*. Vogella. Lars Vogel, 2013. ISBN: 3943747077,9783943747072 (cit. on p. 3).

[YX13] EclipseSource. *Yoxos 5.6 New and Noteworthy*. Blog Entry. EclipseSource GmbH, Mar. 2013. URL: https://eclipsesource.com/blogs/2013/03/15/yoxos-5-6-new-and-noteworthy/ (cit. on p. 11).

All links were last followed on October 18, 2023.

A Code References

This chapter includes all the references to code repositories that were created for this project.

A.1 Palladio Oomph Configuration

This repository contains the Oomph Product files and a product catalog to be able to install Palladio with the Eclipse Installer. The URL is https://github.com/A7exSchin/bootPalladio.

A.2 Eclipse Installation Manager

This repository contains the project for the Eclipse Installation Manager and all its bundles, as well as all releases. The repository ownership has been transferred to the Klibio GmbH organization because they plan to continue development and support for the project. It can be found under https://github.com/klibio/EclipseInstallationManager.

B Tables

Table B.1: Results of the Kano survey in table form. For the answers columns each answer is separated by a semicolon and each position refers to the same person. Red entries are suspected to be faulty or not correctly read by the participant.

Quality Description	Category	Functional Answers				Dysfunctional Answers			
		A1	A2	A3	A4	A1	A2	A3	A4
Mouseless Navigation	Must-be	0	0	++	+	0	0	- -	- -
Shortcut and Tray Application	Indifferent	+	0	++	-	0	0	- -	-
Start installations	Excitement	++	++	- -	+	- -	-	-	-
Show installations	Performance	++	++	- -	++	- -	- -	- -	- -
Delete installations	Excitement	++	++	0	+	- -	-	0	-
Set a workspace for an installation	Indifferent	+	+	0	+	-	-	0	0
Display workspaces	Excitement	++	++	++	+	-	-	- -	-
Set eclipse.ini settings	Indifferent	+	+	0	++	-	0	- -	- -
Configure workspaces	Indifferent	+	+	0	+	-	-	0	-
Configure installations	Indifferent	+	+	- -	++	-	0	0	-
Delete workspaces	Indifferent	+	++	+	+	-	-	0	-
Skip workspace prompt on start	Indifferent	0	+	0	+	0	0	++	-
Add own Eclipse Installations	Indifferent	+	+	0	++	-	0	+	-
Tags for installations and workspaces	Indifferent	+	+	++	0	0	-	0	0
Search installations and workspaces	Performance	++	++	++	+	- -	-	++	-
Hide installations and workspaces	Indifferent	+	+	0	+	-	0	0	0
Include the Eclipse Installation Manager into the Eclipse Installer	Excitement	++	++	++	++	-	-	0	0

Table B.2: This table contains stakeholders that were consulted during the development of the application.

Role	Description	Representative	Availability	Area of profession	Reasons
Customer and User	Wants to use the product for Palladio	Prof. Becker	Via Mail mostly available, else rarely available	Eclipse and Palladio Development	Developers Supervisor, Grades the work
Customer and User	Wants to use the product with the Eclipse IDE	Mr. Kirschner	Via Chat application almost always available	Software Development and Eclipse Development	Supports development with knowledge and ideas
User	Uses product during Palladio development	S. Stieß	Almost everyday in office from 9 - 17, via Mail always available	Palladio Development Process	Example user for the Palladio Development
User	Wants to use the product with his own Eclipse Application	M. Weller	Via Mail mostly available, sometimes in the office	Eclipse Development	Example user for other Eclipse Applications

Table B.3: This table contains all the features that were not implemented and the reason why.

Feature Description	Reason not to implement
Shortcut to open the Tray Application	Problems including the Java Native Libraries into the OSGi framework in addition to the time constraints prevented this feature.
Tags for installations and workspaces	Not important enough, skipped due to time constraints.
Show resource path in the Tray Application when hovering	Not important enough, skipped due to time constraints.
Enable configuration of environmental variables as a launch configuration	Only relevant for a restrictive corporate environment, skipped due to time constraints.
Configure eclipse.ini	Should be properly done in the installations Oomph Product Setup file. Changes locally will be overwritten by the Oomph Setup Core.
Configure workspaces	Should be properly done in the projects Oomph Project Setup file.
Add own Eclipse Installations	Can be done manually, a How-To has been documented[1], but might be implementable with more time.

[1] https://github.com/klibio/EclipseInstallationManager/discussions/35

Table B.4: List of questions asked to participants of the evaluation survey. Each question is listed with the possible answers each participants had.

Question	Answer type
Are there **any problems** with your current development process using Eclipse Applications, which **you see the Eclipse Installation Manager solving**? If yes, please state them.	Long text answer
Are you using or have you used the **Eclipse Installer** before?	Yes/No
Does the **Eclipse Installation Manager** serve as an incentive to use the **Eclipse Installer** for future Eclipse setups?	Yes/No
Rate how the **Eclipse Installation Manager** helps with managing installations/workspaces?	Likert Scale from 1 (Insufficient) to 5 (Exceptional)
Rate the **usefullness** of the **Eclipse Installation Manager**	Likert Scale from 1 (Not useful at all) to 5 (Very useful)
How intuitive is the **user interface** of the **Eclipse Installation Manager** to use?	Likert Scale from 1 (Complex) to 5 (Intuitive)
Rate the **overall usability** of the **Eclipse Installation Manager/**	Likert Scale from 1 (Insufficient) to 5 (Exceptional)
Rate the feedback on critical actions given by the **Eclipse Installation Manager** (like deletion)	Likert Scale from 1 (Insufficient) to 5 (Exceptional)
Rate how the **Eclipse Installation Manager** solves the problem of **managing Eclipse Installations and workspaces**?	Likert Scale from 1 (Insufficient) to 5 (Exceptional)
Did you **miss any options** or settings you would have liked to see?	Long text answer
Rate the **setup process** based on the documentation.	Likert Scale from 1 (Complex) to 5 (Simple)
How do you think the **Eclipse Installation Manager** will influence your development process?	Long text answer
Do you plan to **continue** using the **Eclipse Installation Manager**?	Yes/No
Any additional notes or remarks?	Long text answer

C User-stories

This chapter includes all the user stories in a table format.

User-story C.1

As	User
I want to	be able to see all existing Eclipse IDE installations on my machine.
so that	I can better keep an overview over my development tools
Acceptance criteria	• A view of all available installations • Installations should be identifiable

User-story C.2

As	User
I want to	be able to navigate the applications menus without using the mouse.
so that	move more efficiently through the software
Acceptance criteria	• All menus must be navigable with only using the keyboard • Navigation should stick to common principles, e.g. Tab switches buttons, Enter presses a button

User-story C.3

As	User
I want to	be able to see all existing workspaces on my machine.
so that	I can better keep an overview over my development environment
Acceptance criteria	• A view of all available workspaces • Workspaces should be identifiable

User-story C.4

As	User
I want to	be able to search existing installations and workspaces based on their name
so that	I can find a specific installation or workspace faster
Acceptance criteria	• The views of installations and workspaces includes a search-bar • The search engine searches regular expressions and parts of the names

User-story C.5

As	User
I want to	be able to start any available installation with assigned workspaces
so that	I can use my tools faster and more consistently without the need to search in the file explorer
Acceptance criteria	• The installation view should include a button to launch said installation • The launch should work on multiple machines and operating systems

User-story C.6

As	User
I want to	be able to delete any available installation
so that	I can manage my file system easier
Acceptance criteria	• The installation view should include a button to delete said installation • The deletion should update existing resources, e.g. the location catalog and the applications data

User-story C.7

As	User
I want to	only nned to use one tool to install and manage installations and workspaces
so that	I do not need multiple tools
Acceptance criteria	• Include the Eclipse Installation Manager into the Eclipse Installer. • There should only be one tool needed for installations, management and display of the resources.

User-story C.8

As	User
I want to	name installations and workspaces within the Eclipse Installation Manager
so that	assign a valuable name to installations and workspaces myself.
Acceptance criteria	• Installations and Workspaces should have a name and the capability to change this name. • The changed names should be persistent.

User-story C.9

As	User
I want to	use a global keyboard shortcut to open the Tray Application Main Menu
so that	I can access installations easier and more convenient
Acceptance criteria	• Register a global windows keyboard shortcut that opens the Tray Applications main menu. • The shortcut should not yet be taken. • The shortcut should be configurable.

User-story C.10

As	User
I want to	be able to choose another workspace for an installation that is not yet listed.
so that	create new workspaces or add existing into the Eclipse Installation Manager
Acceptance criteria	• Provide capability to choose or create another workspace for a specific installation.

User-story C.11

As	User
I want to	delete installations and workspaces using the Eclipse Installation Manager
so that	manage installation and workspaces on my hard drive.
Acceptance criteria	• The Eclipse Installation Manager should provider capabilities to delete installations and workspaces. • Deleting an installation or workspace should prompt a confirmation dialog • Deleting installations and workspaces should not result in an inconsistent state of the systems resources.

User-story C.12

As	User
I want to	not be bothered by a workspace prompt, when I start an installation with an assigned workspace
so that	the usage of the Eclipse Installation Manager is more fluent.
Acceptance criteria	• Skip the workspace prompt, when an installation is started with the assigned workspace.

User-story C.13

As	User
I want to	assign tags to installations and workspaces and be able to search for them
so that	customize how the search reacts and improve usability
Acceptance criteria	• Tags should be configurable • The search should react to tags

User-story C.14

As	User
I want to	hide installations and workspaces
so that	reduce the list sizes with installations I do not need right now but do not want to delete either.
Acceptance criteria	• Installations and workspaces should be hidable in the Management View and the Tray Menu. • Hidden items should be in an extra menu to be able to unhide them.

User-story C.15

As	User
I want to	configure the `eclipse.ini` settings of an installations
so that	I am able to tune my installations using the Eclipse Installation Manager
Acceptance criteria	• The user interface should provide means to configure settings inside the `eclipse.ini` file • Changes to the settings should be persistent

User-story C.16

As	User
I want to	configure the `eclipse.ini` settings of an installations
so that	I am able to tune my installations using the Eclipse Installation Manager
Acceptance criteria	• The user interface should provide means to configure settings inside the `eclipse.ini` file • Changes to the settings should be persistent

D Results Data

The specific answers can also be found in the thesis Github Repository[1] as `.xlsx` file. The file also includes text answers. All the results of questions with Likert-scale answers are shown in Figure D.1 on the next page.

[1] https://github.tik.uni-stuttgart.de/st151899/EclipseInstallationManagerThesis

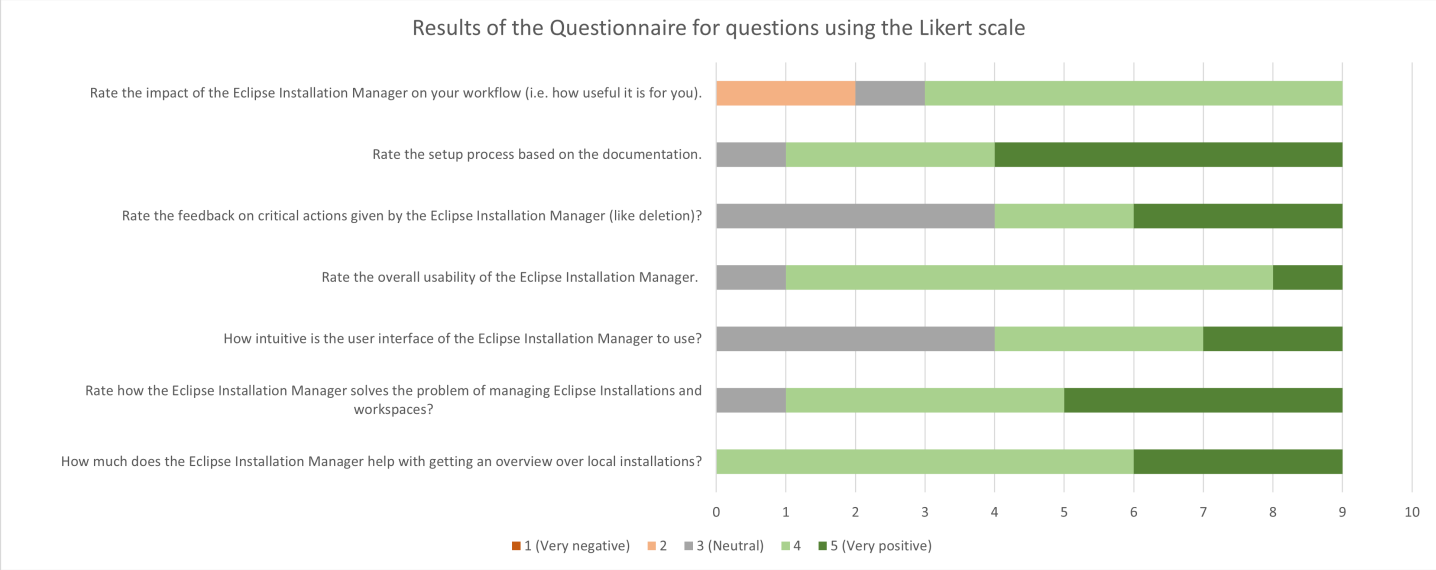


Figure D.1: All results of the questions using a Likert scale. Each bar graph contains all the scores participants gave for a specific question or part of the software.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I 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. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature