SOEN 6471
ADVANCED SOFTWARE ARCHITECTURES
SUMMER 2023

Deliverable 2
ECLIPSE

Declaration

We, the members of the team, have read and understood the Fairness Protocol and the Communal Work Protocol, and agree to abide by the policies therein, without any exception, under any circumstances, whatsoever.

Team K
Preet Angad Singh Nanda
Priyanshi Yogeshkumar Patel
Jimil Suchitkumar Prajapati
Yash Nareshbhai Radadiya
Kevin Rao

Table of Contents

Table of Contents ..... **i**
Problem 4 ..... **1**
List of Quality Attributes that Eclipse IDE Aims to Satisfy 1
List of Quality Attributes that Eclipse Should Satisfy but Does Not 2
Problem 5 ..... **3**
Logical view 3
Use-Case view 4
Deployment view 5
Problem 6 ..... **6**
Desirable Traits 6
Undesirable Traits 7
Problem 7 ..... **10**
Stakeholder and Purpose Orientation 10
Content Quality 10
Presentation and Visualization Quality 10
Management of Documents 10
Responsibilities on Deliverable 2 ..... **11**
Contributions on Deliverable 2 ..... **11**
References **12**

This deliverable submission follows the template provided by [1].

Problem 4

List of Quality Attributes that Eclipse IDE Aims to Satisfy

1. **Modularity**:

The degree to which a product is composed of discrete modules such that a change to one module has minimal impact on other modules [14].
Eclipse's architecture is designed to support modularity, allowing developers to build and extend the platform with reusable components and customize the IDE with plugins and add-ons [13].

2. **Interoperability**:

The level of compatibility and interoperability between multiple software products, allowing them to share information and effectively utilize the exchanged data [14].
Eclipse supports integration with other tools and frameworks, emphasizing interoperability through its framework (OSGi), enabling developers to work seamlessly with external resources and systems [13].

3. **Reliability**:

The extent to which a software product executes predetermined tasks, within predetermined circumstances, over a predetermined duration [14].
Eclipse guarantees reliability through the ALF framework, which aims to coordinate distributed tools [11].
Reliability refers to the extent to which a product successfully carries out its intended functions within predetermined conditions and for a specific duration. It is one of the initial quality factors that end-users are likely to observe. This attribute is evaluated based on two key aspects: the frequency of problems encountered and the intricacy of the underlying code which are evaluated and satisfied by Eclipse [12].

4. **Extensibility**:

Extensibility is a fundamental principle in software engineering and systems design that focuses on accommodating future expansion. It refers to the system's capacity to be easily extended and the amount of effort needed to implement those extensions. These extensions can involve introducing new features or modifying existing ones. This principle allows for system enhancements without negatively affecting its current functions [15].
Eclipse provides a plugin-based architecture that allows developers to customize and extend the functionality of the IDE, enabling support for various programming languages, frameworks, and tools [16].

5. **Scalability**:

The ability of a system to manage a greater or lesser amount of work by adjusting the cost of the system. By allowing the selective inclusion of essential components, the modular approach enhances the scalability of Eclipse, reducing resource usage and enhancing performance [14].
Eclipse provides various mechanisms for distributed development and collaboration, such as support for version control systems and team development tools. These features enhance scalability by allowing multiple developers to work on the same project simultaneously and facilitating efficient code management and collaboration in larger development teams [13].

6. **Maintainability**:

The level of effectiveness and efficiency in modifying a software product by the designated maintainers [14].
Eclipse has a large community of contributors and users who work together to improve and enhance the platform. Additionally, the use of a modular architecture and the adoption of OSGi as a component model that contribute to the maintainability of the Eclipse platform [18].

7. **Reusability**:

The degree to which an asset can be used in more than one software product, or inbuilding other assets [14].
Eclipse's architecture is designed to support modularity, allowing developers to build and extend the platform with reusable components [13].

8. **Accessibility**:

The extent to which a software product can be utilized by individuals with diverse characteristics and abilities to accomplish a specific objective within a given context of use [14].
Eclipse incorporates various accessibility features to aid individuals with disabilities or special needs in utilizing the software effectively. These features include keyboard-based operation, compatibility with screen-reader software and speech synthesizers, as well as the ability to magnify graphical views on the screen [16].

List of Quality Attributes that Eclipse Should Satisfy but Does Not

1. **Security**:

The software product's ability to protect information from being read or altered by unauthorized individuals or systems, while ensuring authorized individuals or systems can access it without any restrictions [14].
Eclipse primarily focuses on development features and may not provide robust security measures to protect against vulnerabilities or code injection attacks. Strengthening

security practices, such as code scanning and vulnerability analysis, could improve this aspect.

2. **Customizability**:

The ability to be customized [15].
While Eclipse offers extensibility through plugins, it may lack a flexible and user-friendly interface for customizing the IDE's appearance and layout. Providing more options for UI customization would allow developers to tailor the environment to their preferences.

3. **Compatibility**:

A software product's capacity to share data with other software products and/or carry out its necessary operations while utilizing the same hardware or software environment [14].
Eclipse should support multiple operating systems and integrate well with other development tools and frameworks, but currently supports Windows, Linux and macOS only.
Eclipse traditionally focuses on desktop and web development, and its support for mobile platforms may not be as comprehensive as specialized mobile IDEs. Expanding mobile development capabilities and providing better integration with mobile SDKs could address this limitation.

Problem 5

Orthogonal views are a very important part of describing software architecture. Oftentimes, the software requirements are themselves too complex and multi-layered to begin with. Views highlight various concerns at different levels of abstraction, which helps stakeholder and developers to understand the software.
These different views enhance communication between developers and stakeholders. Moreover, they manage the complexity of the project, improve documentation by adding visual elements and also aid in decision making by analyzing different types of views.

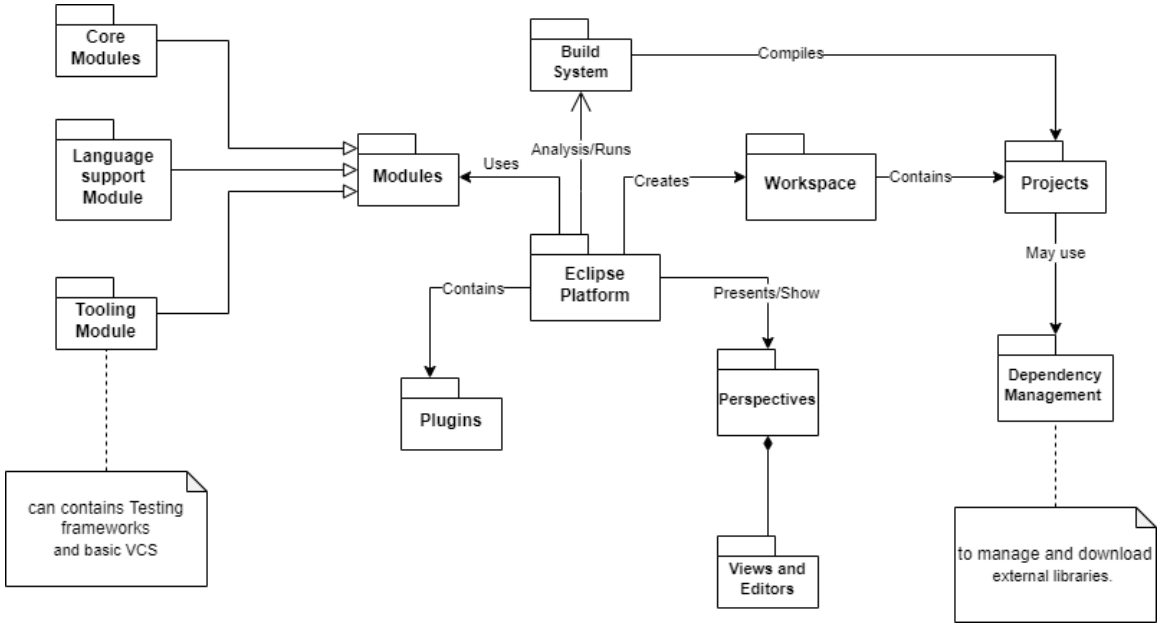
Here, for understanding architecture of Eclipse, we looked at three important views which highlight different aspects of the system and help to see the broader picture.

These views mainly came from **“The 4+1 View Model of Software Architecture”**.[7]
By choosing the established viewpoint model we can have less ambiguity in our views and can have better documentation and support from the community.

Logical view

Logical view focuses on functionality and structure of the problem domain at mainly higher levels of abstraction.[6] It will help show how components interact with others and

also showcase the functionality and/or capabilities provided by eclipse, e.g. editing, debugging, VCS, etc.



Modules - Modules are self-contained units within Eclipse IDE. Unlike plugins, modules come with the IDE and are very important for working of the software as they contain various small resources, like programming language support, UI rendering and many more.

Perspective - It organizes different views (e.g. console, code editor, working file tree) into the screen and presents it to the screen.

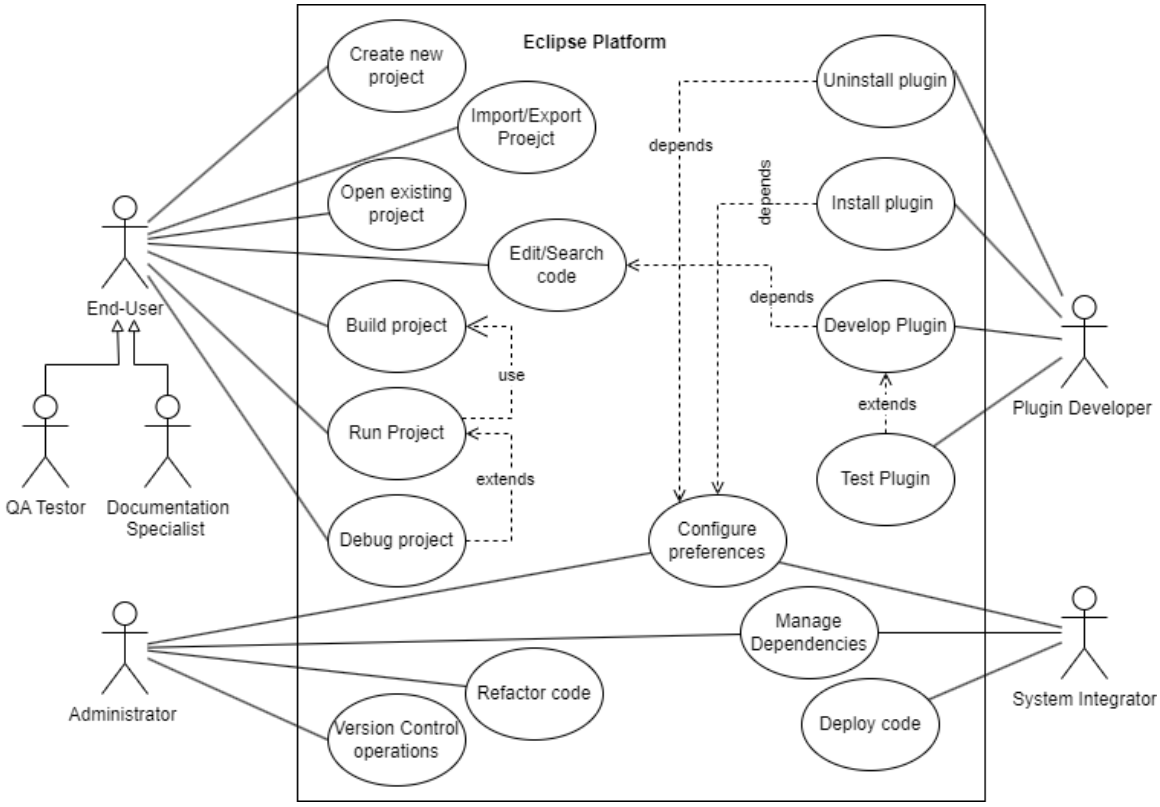
Workspaces - It is a central concept in Eclipse, which is responsible for location of different projects (contains source code and resource files) and their resources.

Build System - To compile and build the project files. Often includes some form of automation tools and different build configurations. it may be different based upon programming languages used.

Use-Case view

The said view is considered to be very important, for it is responsible; how the developed software is perceived by the end users. While other views presented here mainly focus on the internal structure, the use-case view exposes external functionality. It's useful because it can provide insights into the interaction between software and end-users.

Moreover, the use-case view is rather useful when it comes to the validation phase. It can serve as a requirement validation and verification, and by reviewing use-cases stakeholders can ensure that all the necessary functionalities are included and working as intended.



End-User - The person who will use Eclipse to develop the software. Here, the end-user is not generalized as every person using Eclipse to do their task, but is limited to mostly low level development (such as writing code a major part).

Administrator - The person who is responsible for managing Eclipse installations and configuration. Most large scale projects include an admin who mainly handles the system surrounding the software environment.

Plugin Developer - They're essential to the productivity of the developers as they mainly automates repetitive tasks, which saves much time and effort, and can introduce small pieces of mini-resources to help the programmers with their tasks.

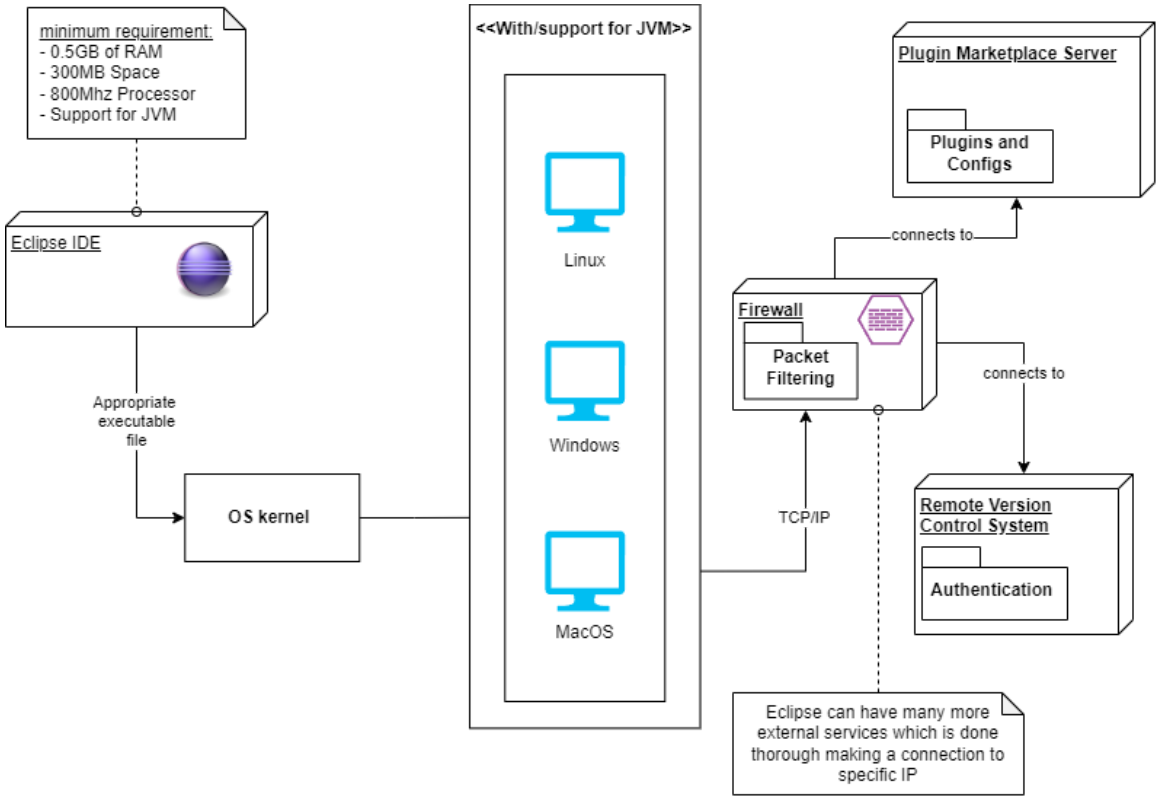
System Integrator - System integrator is responsible for integrating Eclipse with other softwares in the system and can help with deployment of the software and dealing with dependencies to manage resources.

Deployment view

Deployment view mainly focuses on physical deployment of the software system onto hardware infrastructure whether it be an individual machine, server or hybrid medium. Deployment also involves installation, configuration and setup of a system to make sure it performs well and smoothly.

This view helps the stakeholders understand how the system is deployed and assist them to judge its reliability, scalability and availability.

Any large software is fairly complex and comes with its own criteria to ensure it works in a particular environment. Thus, creating a deployment view allows us to visualize the deployment architecture and how software interacts with the surrounding hardware.



The above deployment view depicts how Eclipse can run on a particular system and communicate with remote entities. The minimum requirements[8] to run the software is substantially modest which makes it easier for end user to use the software smoothly.

Problem 6

There are many patterns, style principles and tactics present in Eclipse's software architecture. There are also undesirable traits as well. The following subsections list these out.

Desirable Traits

Modular Architecture: Eclipse's architecture is based on a modular design, allowing developers to add or remove specific features through plugins. This modular approach enables flexibility and extensibility, making Eclipse adaptable to different programming languages and development workflows.

Service-Oriented Architecture (SOA): Eclipse's architecture is built upon the Eclipse Platform, which provides a collection of essential services and frameworks. These

services can be accessed and utilized by plugins, promoting a service-oriented architecture that encourages modularity and reusability.

Model-View-Controller (MVC) Pattern: Eclipse has the Model-View-Controller pattern to separate concerns and enhance maintainability. The code editors, integrated debugging tools, and other visual components of Eclipse can be seen as views, while the underlying code and data models represent the model. The controller handles the interactions between the view and model components. [9]

Plugin Architecture: Eclipse utilizes a plugin-based architecture, where developers can extend the functionality of the IDE by creating and integrating plugins. This approach promotes code modularity, reusability, and customization, allowing users to tailor Eclipse to their specific needs.

Layered Architecture: Eclipse may employ a layered architecture, where different components and services are organized into layers, each responsible for specific functionalities. For example, there could be layers for user interface, language support, project management, and debugging, allowing for separation of concerns and promoting maintainability. [9]

Event-Driven Architecture: Eclipse follows an event-driven architecture, where various components and plugins can subscribe to and emit events. This enables loose coupling between components, facilitating extensibility and modularity.[10]

Continuous Integration and Delivery (CI/CD): Eclipse supports integration with external tools and frameworks, such as version control systems and testing frameworks. This integration aligns with the principles of continuous integration and delivery, allowing developers to seamlessly incorporate these tools into their development workflow.

Separation of Concerns (SoC) Principle: Eclipse emphasizes the separation of concerns, dividing the software system into distinct modules or plugins. Each module focuses on a specific functionality, promoting modularity, maintainability, and code reusability.[10]

Dependency Injection (DI) and Inversion of Control (IoC): These patterns facilitate loose coupling, testability, and flexibility in managing component dependencies.

Undesirable Traits

A trait, pattern, or part that is deemed harmful or unwanted for the overall quality and efficacy of the architecture can be referred to as undesirable in a software architecture. These undesirables can make the software architecture unstable, unsupportive, and/or ugly.[5]

Undesirables may appear as architectural odours, anti-patterns, or design defects, among other manifestations. In the context of the Eclipse software architecture, we refer to odours as potential design flaws or areas for development. These odours can be used to

spot architectural flaws that might impair the Eclipse system's ability to be scaled, maintained, or otherwise be of high quality.

Here are some common smells in Eclipse software architecture:

1. Rigidity: The design is difficult to change[5]. In other words, Rigidity is a smell that suggests the architecture is inflexible and resistant to change. In Eclipse, this can be observed when modifying or extending existing functionality becomes challenging due to tightly coupled components or lack of proper abstractions. Knowledge of this smell can prompt developers to prioritize modularity, loose coupling, and well-defined interfaces, allowing for easier adaptation and evolution of the system.

2. Fragility: The design is brittle[5]. Fragility refers to the tendency of the system to break in unexpected ways when changes are made. In Eclipse, this smell can arise when modifying one part of the system inadvertently affects unrelated components due to hidden dependencies or improper isolation. Recognizing this smell can encourage developers to apply principles of encapsulation, loose coupling, and thorough testing to minimize unintended consequences and improve overall system robustness.

3. Needless Complexity: There is overdesign or ephemeral complexity[5]. Needless complexity refers to architectural designs that are overly complicated, introducing unnecessary intricacies and making the system harder to understand and maintain. While Eclipse is a complex platform, it provides mechanisms and guidelines to manage complexity effectively. However, needless complexity can still arise if developers do not follow architectural best practices or if there is inadequate documentation. It is essential to maintain simplicity, use clear abstractions, and adhere to modular design principles to avoid unnecessary complexity.

4. Needless Redundancy: There is unmanaged redundancy[5]. Needless redundancy occurs when multiple components or functionalities perform similar or identical tasks, resulting in duplicated effort and increased maintenance overhead. In Eclipse, this smell can emerge if developers create plugins or extensions that duplicate existing functionality without a clear justification. It's important to ensure proper coordination and collaboration among developers to avoid redundancy and encourage code reuse when appropriate

5. Opacity: Opacity refers to a lack of clarity and transparency in the architecture, making it difficult to understand the system's behavior and dependencies. Eclipse, being an open-source platform with extensive documentation and a strong developer community, actively encourages transparency and provides resources to understand the architecture and its components. However, if there is a lack of documentation or if certain components are poorly documented or lack clear interfaces, opacity smells can still occur. Maintaining clear documentation and consistent communication can help mitigate opacity and improve the understandability of the architecture.

Use of knowing the undesirables in a Software Architecture

1. Identification of Design Issues: It helps developers and architects identify potential design issues or weaknesses in the architecture. By recognizing these smells, they can proactively address them, improving the overall quality of the software system.

2. Maintenance and Evolution: smells often indicate areas of the architecture that are difficult to maintain or modify. By being aware of these smells, developers can prioritize refactoring efforts, making the codebase more maintainable, extensible, and adaptable to future changes.

3. Quality Assurance: When conducting code reviews or quality assurance activities, knowledge of architectural smells enables developers to identify and provide feedback on potential issues early in the development process. It promotes better code quality and adherence to architectural best practices.

4. Scalability: Smells related to performance bottlenecks or inefficiencies in the architecture can guide developers in optimizing critical paths and ensuring the system meets performance requirements. Understanding these smells helps in identifying and addressing potential scalability issues as well.

5. Collaboration: it enhances collaboration among developers and architects. By having a shared understanding of common smells, team members can effectively discuss architectural decisions, identify areas for improvement, and communicate the rationale behind refactoring efforts.

6. Learning and Growth: Understanding smells in the architecture of Eclipse provides a valuable learning opportunity for developers. By analyzing and addressing these smells, developers gain experience in architectural design principles, software engineering best practices, and the nuances of building complex software systems.

In conclusion, understanding the Eclipse architecture's features aids in enhancing code quality, maintainability, and overall system performance. Developers can use it to optimize the architecture for scalability, encourage productive teamwork, and proactively detect and fix design concerns. In the end, this information aids in leveraging the Eclipse framework to create software applications that are more reliable, expandable, and effective.

Problem 7

As per [2], there are four major aspects that can be improved in documenting software architecture. Many of which are arguably relevant to Eclipse. The following subsections list and explain the significance of these aspects in improving the software architecture and rely on the information presented in [2].

Stakeholder and Purpose Orientation

Eclipse should be aware of who is interested in the software and update or remove accordingly as the software evolves. The finding of the relevant stakeholders can be guided by keeping in mind the purpose Eclipse intends to fulfill, and by and for whom the purpose serves. Defining the stakeholder would allow Eclipse to be more cognizant and targeted as to how it fills each stakeholder's needs and demands in the market.

Content Quality

Documentation regarding Eclipse's software architecture should be consistent, i.e. contain no contradictions. This is important to avoid confusion and ambiguity in the reader. Moreover, it allows one to more confidently combine the information presented from the various viewpoints across the documentation together for a more holistic understanding of Eclipse as a whole. To support this, thorough information that leaves no stone unturned would leave no possible question unanswered. As well as ascertaining that the documents are up to date and match the reality of the Eclipse software.

Presentation and Visualization Quality

To ensure the software architecture of Eclipse is readily understandable, there are a number of elements to be mindful of. Establishing and conforming to standards, as well as explaining in a clear and intuitive manner allow the community to easily learn the architecture with as few hurdles as possible. In addition, lowering the complexity of the architecture description by abstraction and generalization can help readers track all the elements and their relationship by minimizing the maximum number of interactions between elements at once. Diagramming and visual presentation can be improved by following some Gestalt principles, such as those for proximity, common fate, closure and similarity[3], increasing the readability and presentation value.

Management of Documents

Increase the process quality to uphold the software architecture quality over time. This is possible by having the documentation be maintainable (such as tracking a history of revisions), cost effective and by leveraging the use of tools to streamline and ease documentation.

Responsibilities on Deliverable 2

Team Member	Responsibilities
Preet Angad Singh Nanda	● Problem 6
Priyanshi Yogeshkumar Patel	● Problem 4
Jimil Suchitkumar Prajapati	● Problem 5
Yash Nareshbhai Radadiya	● Problem 6
Kevin Rao	● Problem 7

Contributions on Deliverable 2

Team Member	Contributions
Preet Angad Singh Nanda	● List of undesirables in the software architecture of Eclipse ● Uses of such architectural Knowledge ● Review documentation
Priyanshi Yogeshkumar Patel	● List the quality attributes Eclipse satisfy ● List and gave reasonable comments on quality attributes that Eclipse doesn't satisfy ● Review the documentation
Jimil Suchitkumar Prajapati	● Explained three orthogonal views including three UMLs diagrams. ● Reviewed and edited the deliverable 2.
Yash Nareshbhai Radadiya	● List of patterns, principles, styles, and/or tactics of software architecture used by Eclipse. ● Use of such architectural knowledge
Kevin Rao	● Setup shared collaborative Google Docs and page numbering, filled cover page of template, cited the template. ● Problem 7, avenues of improvements. ● Helped review view diagrams

References

[1] Kamthan, P. (2023) Documentation Template [Word Document]. Course Web Site. http://users.encs.concordia.ca/~kamthan/courses/soen-6471/documentation_template.docx

[2] Hämäläinen, N., Markkula, J. Question framework for architectural description quality evaluation. Software Qual J 17, 215–228 (2009). https://doi.org/10.1007/s11219-008-9068-1

[3] Todorovic, Dejan. "Gestalt principles." Scholarpedia 3, no. 12 (2008): 5345. doi:10.4249/scholarpedia.5345

[4] Brunet, J., Murphy, G. C., Serey, D., & Figueiredo, J. (2014). Five years of software architecture checking: A case study of Eclipse. *IEEE Software*, *32*(5), 30-36.

[5] Kamthan. P. (2023) The 'undesirables' of Software Architecture https://users.encs.concordia.ca/~kamthan/courses/soen-6471/software_architecture_undesirables.pdf

[6] P. Kamthan, VIEWS AND VIEWPOINTS OF SOFTWARE ARCHITECTURE, https://users.encs.concordia.ca/~kamthan/courses/soen-6471/software_architecture_views_viewpoints.pdf

[7] Wikipedia "4+1 architectural view model," Wikipedia, https://en.wikipedia.org/wiki/4%2B1_architectural_view_model

[8] T. Team, "Intellij idea vs eclipse: Which is better for Beginners," The Official Tabnine Blog, https://www.tabnine.com/blog/intellij-idea-vs-eclipse

[9] Richards, Mark. Software architecture patterns. Vol. 4. 1005 Gravenstein Highway North, Sebastopol, CA 95472: O'Reilly Media, Incorporated, 2015 https://isip.piconepress.com/courses/temple/ece_1111/resources/articles/20211201_software_architecture_patterns.pdf

[10] https://www.ou.nl/documents/40554/791670/IM0203_03.pdf/30dae517-691e-b3c7-22ed-a55ad27726d6

[11] ALF/architecture/ALF Reliability https://wiki.eclipse.org/ALF/architecture/ALF_Reliability

[12] An Empirical Study on the Maturity of the Eclipse Modeling Ecosystem https://www.researchgate.net/publication/319208818_An_Empirical_Study_on_the_Maturity_of_the_Eclipse_Modeling_Ecosystem

[13] The Architecture of Open Source Applications (Volume 1) Eclipse https://aosabook.org/en/v1/eclipse.html

[14] P. Kamthan, INTRODUCTION TO SOFTWARE PRODUCT QUALITY, https://users.encs.concordia.ca/~kamthan/courses/soen6471/software_product_quality_introduction.pdf

[15] List of system quality attributes https://en.wikipedia.org/wiki/List_of_system_quality_attributes

[16] Help - Eclipse Platform https://help.eclipse.org/latest/index.jsp

[17] https://wiki.eclipse.org/EclipseQualityModel

[18] https://wiki.eclipse.org/EclipseQualityRequirements