Conceptual Architecture Description and Evaluation Report (T2)

Delivery Date: Jan 4, 2024

Team Members:

Ahmet Ensar Sönmez 21827794

Eyüp Zafer Ünal 21727848

Fevzi Kılas 2200356822

1. OSS PRODUCT OVERVIEW

Name of OSS Product:	FreeCol
Description of OSS product:	FreeCol is a turn-based strategy game based on the old game Colonization, and similar to Civilization. The objective of the game is to create an independent nation.
URL of OSS product:	https://github.com/FreeCol/freecol
Size of OSS product (KLOC):	182K

Table 1. Description of the OSS product.

2. CONTEXT DIAGRAM

FreeCol is a Rich Client Application. Rich Client Applications are installed and run on a user's machine. So applications run on the users machine and this gives us higher performance, interactive and rich user experience. Since FreeCol is an online game, this decision is well chosen.

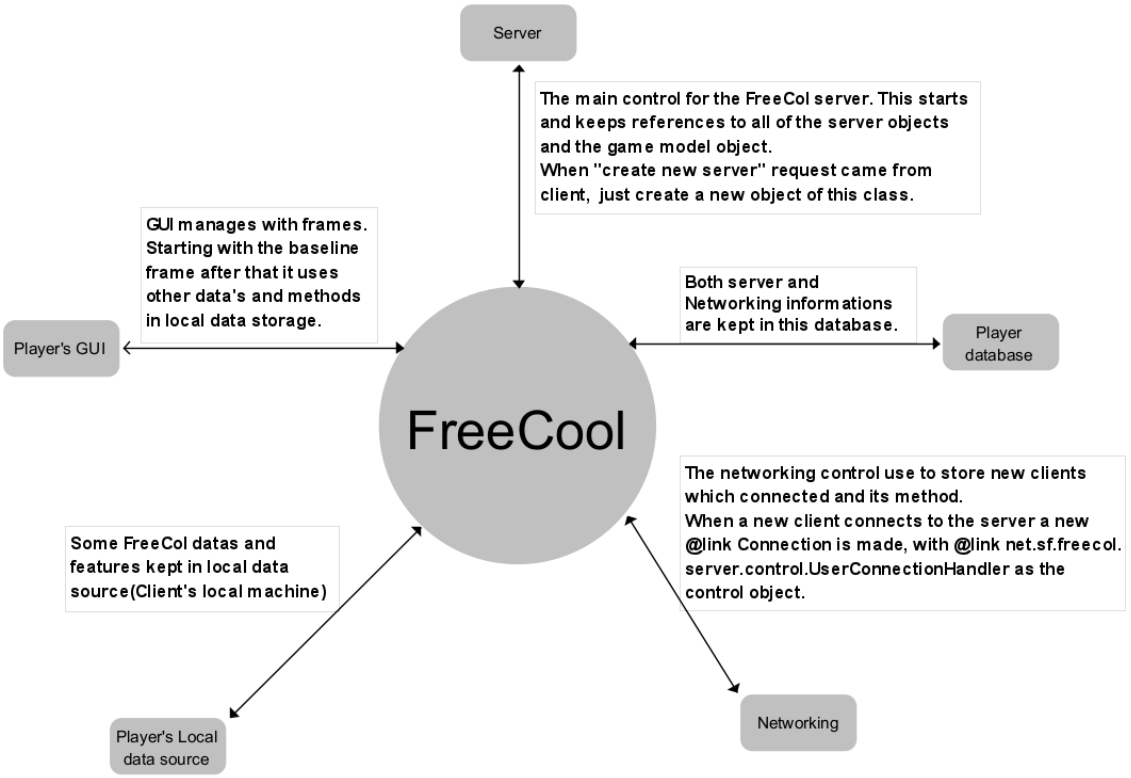


Table 2. CONTEXT DIAGRAM of the OSS product

3. CONCRETE ARCHITECTURE EVALUATION SUMMARY

Let's have a look at the conclusions about T1. The first thing which I have to mention is complexity. The project FreeCol is generally a complex software according to metrics we obtained. Some classes have very high complexity. We have also seen reusing some code parts by inheritance. Dependencies between classes are also high, which means that code maintainability is a bit difficult. Single responsibility principle isn't commonly used in practice as we see. There are a lot of useful comments that help you while reading the code.

Our opinions about the project FreeCol are that it could have been better according to metrics we obtained. They could have reduced the complexity of the classes and reform some of the classes to fit the single responsibility principle. The best thing about this project is code comments. There are so many proper comments that explain in the simplest way.

4. TOOLS USED FOR CONCEPTUAL ARCHITECTURE DESCRIPTION

Tool Name	Purpose of Use	URL
Visual Paradigm	*We used for creating diagrams*	https://www.visual-paradigm.com/
Understand	We used for creating dependency graphs	https://scitools.com

Table 3. The list of tools used for architectural description

5. DESCRIPTION OF MODULE VIEW(S) fevzi

5.1. Module View

For the module view of FreeCol we investigate the packages carefully. We mainly focused on source code. Source code of the system is in the path src/net/sf/freecol. We show these folders using UML Package Diagrams because it focuses on organizing files in an object oriented way. Also to see structural elements more clearly. So our system runs on **FreeCol.java** file and it imports all the packages and starts our system. We show the module view in Figure 1.

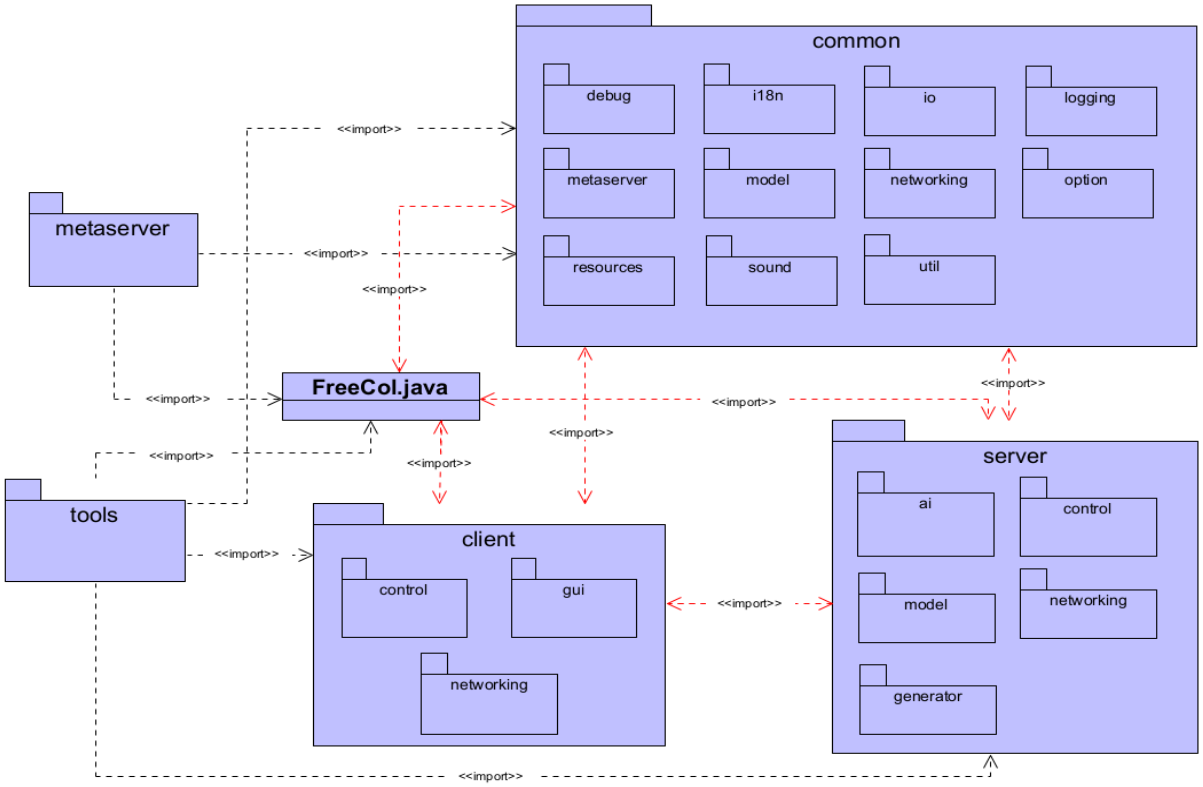


Figure 1: Module view for the OSS product

Common module contains packages in use by both the server and the client. Therefore its name is common. It also contains debugging packages for exception handling. It also contains these packages;
- classes for handling logging information within FreeCol.
- common networking classes
- game model, which describes how the individual game objects, such as units, buildings, tiles…

- classes for describing, and saving the state of a game options
- classes for handling resources

- classes for handling sound effects and music in FreeCol

Client module contains FreeColClient.java file and it is the main control class for the FreeCol client. This class references GUI and controls objects. Client module also has control, gui and networking packages in it. Basically the control package contains the packages responsible for the control of the game. The GUI package contains the graphical user interface classes. And finally the networking package contains the client networking files.
Server module managed by FreeColServer.java and it is the main control class for the FreeCol server and responsible for network communication. It also has some useful packages. These packages is;
- Ai is the main package of the Artificial Intelligence (AI) package tree.
- Control contains the classes responsible for the control of the game.
- Generator contains the map generator.
- Model contains model classes with server specific information.
- Networking contains the server networking classes.

Other packages are **tools** and **metaserver**. **Tools** contains tools directly related to the FreeCol game itself for example; ColonizationMapReader. **Metaserver** contains an implementation of a meta server.

5.n. Traceability of the Module View to the Dependency Graph

In the T1 part our dependency graph has a missing part which is a test package. We added an updated dependency graph to the appendix part. Also we added an attachment for the package level dependency graph because it is too big for this pdf file. You can easily see that in the dependency graph (graph x) all the modules exist perfectly. We examined all modules and their packages. We could not find more modules or structures to add the module view. So we can say that the product team did a great job for this subject.

Module View (number or name)	Dependency Graph (number or name)
Figure 1: module view	Graph 1 (in the appendix)

Table 4. Traceability of the module view(s) to the dependency graph(s)

6. DESCRIPTION OF COMPONENT AND CONNECTOR (C&C) VIEW(S)

We want to show how the game works in terms of inputs, output and behind jobs. So you can see in Figure 2 we try to represent this scenario. We will explain each component and the general logic behind it but before that we want to say that this c&c view gives us a graphical representation of the system components but for this specific scenario so we couldn't use all the components.

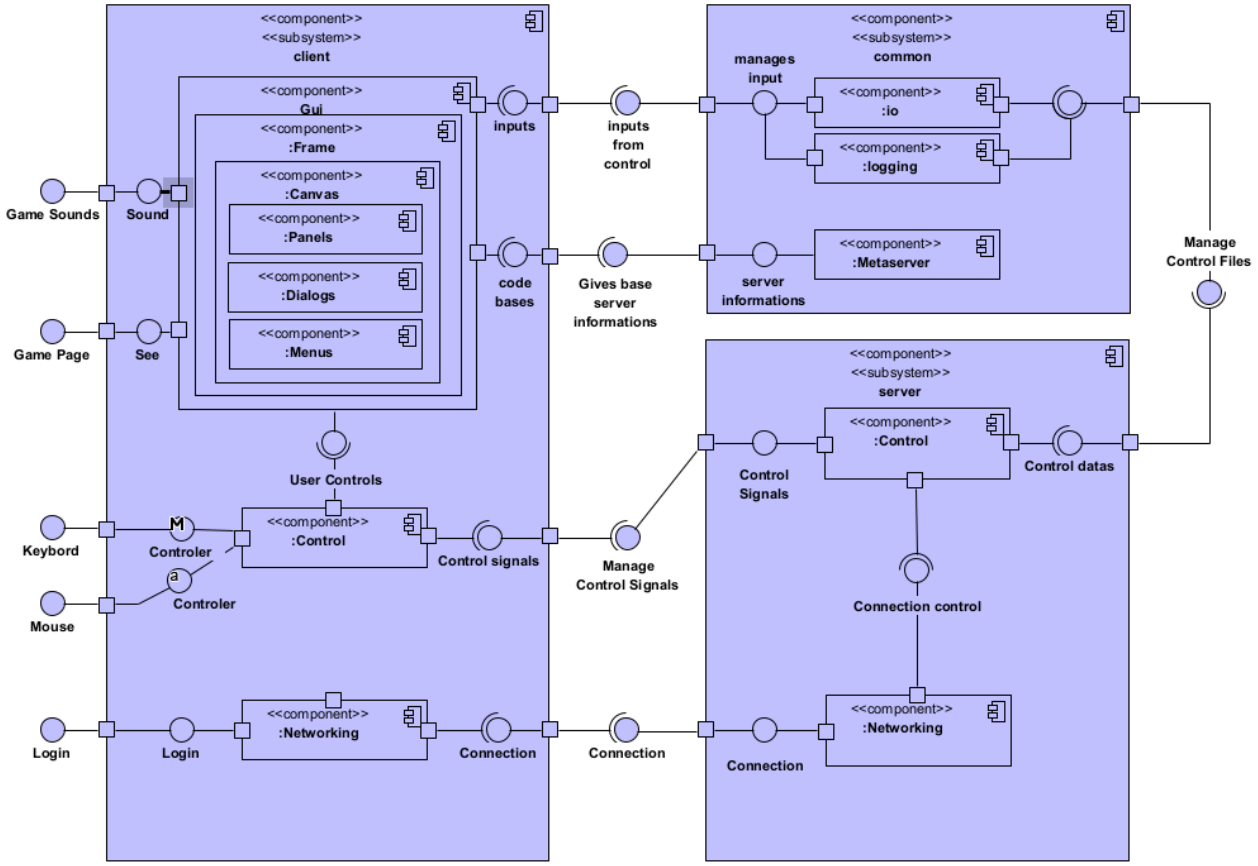


Figure 2: C&C view of the OSS product

Client component contains Gui, Control and Networking sub components. Gui is the graphical user interface component and it is used for the output of the system. First of all, the player logins the system and then creates a server. When this server created Frame object is created automatically. Frame component handles both windowed and full screen presentations and some other things. After that Canvas object is created and it contains in-game pictures or sounds or menus. Player controls these components with the controller component in the client component. Since this FreeCol game is an online game players actions should affect other players. For this kind of scenario control components interact with the server. In the server the request from the client is handled and then it is stored or what needs to be done is done. And also the login part handled in the server since passwords and usernames are stored on the server. Finally metaserve is used in the beginning when the frame is created. It contains predetermined information, images or sounds for the starting map or chosen map from the client.

Component(s) in C&C view(s)	**Element(s) in module view(s)**
Client	client
Gui	gui
:Frame	not represented
:Canvas	not represented
:Panel	not represented
:Dialogs	not represented
:Menus	not represented
:Control	control
:Networking	networking
Common	common
:io	io

:logging	logging
:Metaserver	metaserver
server	server
:Control	control
:Networking	networking

Table 5. Traceability of the components in the C&C view(s) to the elements in the module view(s).

7. DESCRIPTION OF ALLOCATION VIEW

Allocation structures show the relationship between the software elements and elements in one or more external environments in which the software is created and executed. We do not have too much information about freeCol databases and ai so these elements are just shown without details.

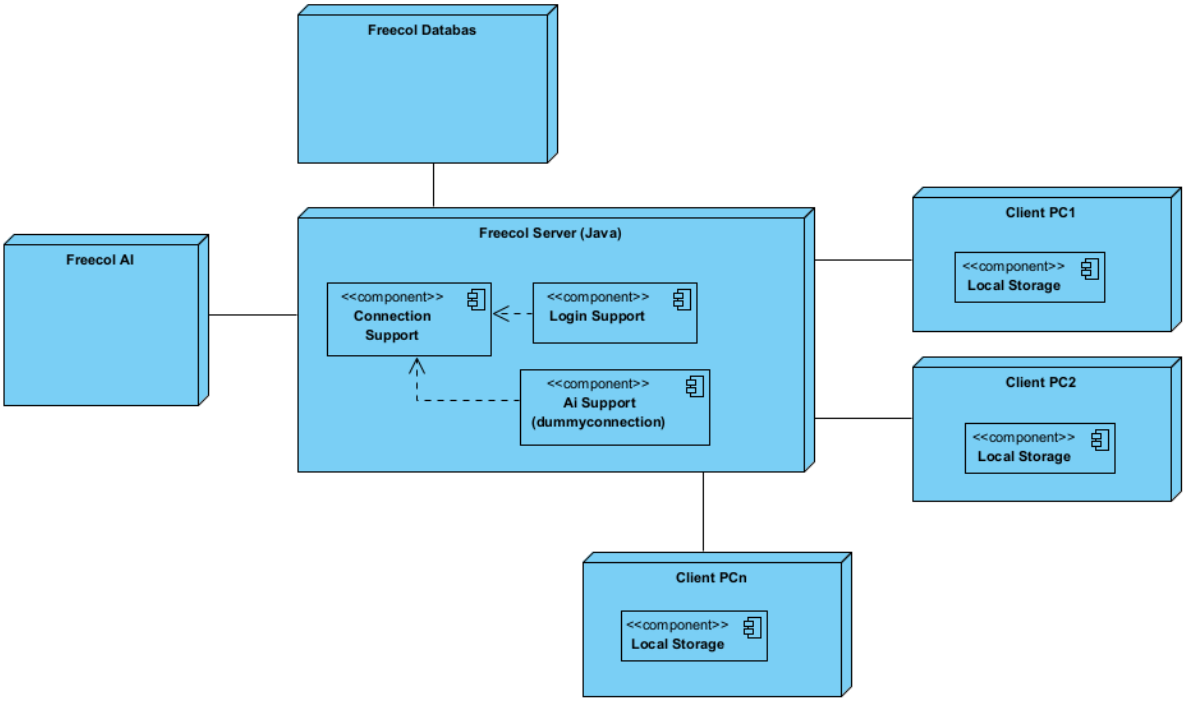


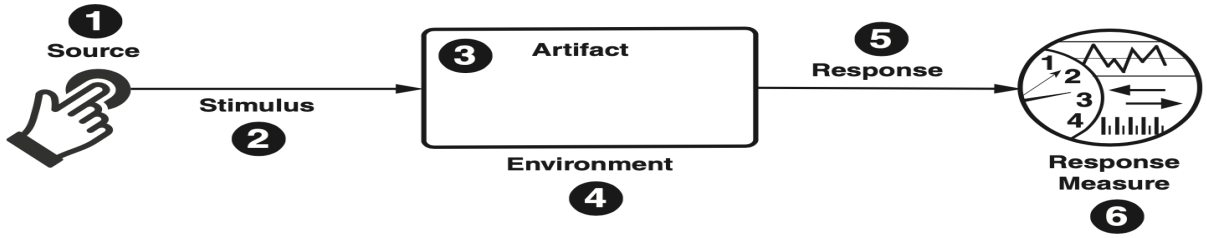
Figure 3: Allocation view of the OSS product

Database responsible for storing servers's address, client's usernames and passwords, and other database management stuff.
FreeCol AI responsible for some game features and in-game bots. Bots create dummy connections and they have some special features and movements.
FreeCol server is responsible for the online connection between players and it manages game servers. Server also responsible for login support and ai support.

Finally clients represent players. Each player runs the game in their local machines and these machines have their own storage. Game files stored in those storages. If a player wants to login the game it sends a request to the server. In the same way if a player creates or joins a server it sends a request to the server.

8. DESCRIPTION OF SPECIFIC QUALITY ATTRIBUTE (QA) SCENARIO

We use a common form to specify all quality attribute requirements as scenarios. We will explain parts of the quality attribute requirements one by one.



1. Source of stimulus: A software defect is discovered in a critical system functionality.
2. Stimulus: The development team needs to fix the defect and verify the fix without affecting other functionalities.
3. Environment: The system is in production, serving its normal workload.
4. Artifact: The system's source code, including unit tests, integration tests, and deployment scripts.
5. Response: The development team identifies the root cause of the defect, fixes it in a, verifies the fix through automated and manual testing, and deploys the fix to production with minimal downtime and impact on users.
6. Response measure:
    a. Time to locate and fix the defect (measured in hours).
    b. Number of additional defects introduced during the fix (measured as 0).
    c. System downtime during deployment (measured in minutes).
    d. User feedback on functionality and stability after deployment (measured through surveys or monitoring).

We choose this specific quality attribute scenario but first we need to understand the differences between general and specific quality attribute scenarios. General quality attribute scenarios are system independent and can, potentially, pertain to any system but concrete scenarios are specific to the particular system under consideration. For instance, a specific scenario may be login failure but its general quality attribute scenario should be failure in the system functionality.

In this scenario we detect defects in critical system functionality and the software team should fix it. The key here is that these bugs must be fixed without affecting other functionalities. To do this the development team identifies the main reason behind this defect and tries to fix the defect with minimal downtime and localized manner(not affect other functionalities).

Finally we need to measure these responses of the developers. We can look these questions to measure;
How fast is the bug fixed?
What happened during the fix?
Did the fix affect the program?
How many hours was the system shutdown during the fix?

Specific QA scenario	Question in GQM tree	Explanation (or rationale)
Defect in the critical system functionality.	Q9- How often are complex algorithms or calculations handled within a class?	More the complex system more likely trouble in other functionalities or more hard to fix defect by development teams.

Table 6. Traceability of the specific quality attribute scenario to the question and answer in GQM tree

9. SUGGESTIONS FOR IMPROVEMENT OF THE PRODUCT ARCHITECTURE

9.1 Suggestion for Complexity of the Product

As we all know, high complexity is unacceptable for software products. High complexity has bad implications for product development and maintenance. Poorly managed projects often experience high complexity. Complexity also makes things harder when it comes to detecting and fixing bugs. Reducing this complexity not only facilitates the management of the project, but also increases the quality and sustainability of the product.

In this particular project, the complexity seems high according to the metrics we obtained. It is possible to solve this complexity in some ways that I will list below.

1. To make classes simpler and more readable by reviewing class structures and removing unnecessary or obsolete structures.

2. Separating interconnected classes from these dependencies as much as possible and making classes and methods comply with the single responsibility principle

9.2 Suggestion for Readability of the Product

A good software product is also one that is readable and understandable. Readability not only makes the project more organized, but also makes the work of developers working on the project easier. One consequence of the high complexity in this project is that it reduces readability. Although this project provides us with explanations with comment lines, the readability of the code has decreased due to the complexity. Our suggestions to reduce the complexity in order to solve this problem can be found in recommendation 9.1. At the same time, we will have some more suggestions to increase readability. Editing the file structure of the project will make it easier for us to access some files on the project and we will have a more organized project.

Long lines of code also pose a problem that reduces readability. Code lines exceeding a certain number of characters should be divided into pieces as much as possible to increase the readability of the code.

Another problem we encounter in this regard is having long methods that take too many arguments or do not comply with the single responsibility principle. If a method takes more than 2 or 3 arguments, this automatically reduces the readability of that method and makes it difficult for us to understand what it does. If the method has more than one task, naturally more code is written on the method and readability decreases.

Our last suggestion about readability will be about naming. When the naming of variables, methods and classes is done carefully, the code becomes more readable. These nomenclatures should be short and clear and should give us a brief idea of what that structure is. At the same time, creating harmony within the project by naming in accordance with the conventions of the programming language used or the rules determined for the project helps us to increase readability.

10. REFERENCES

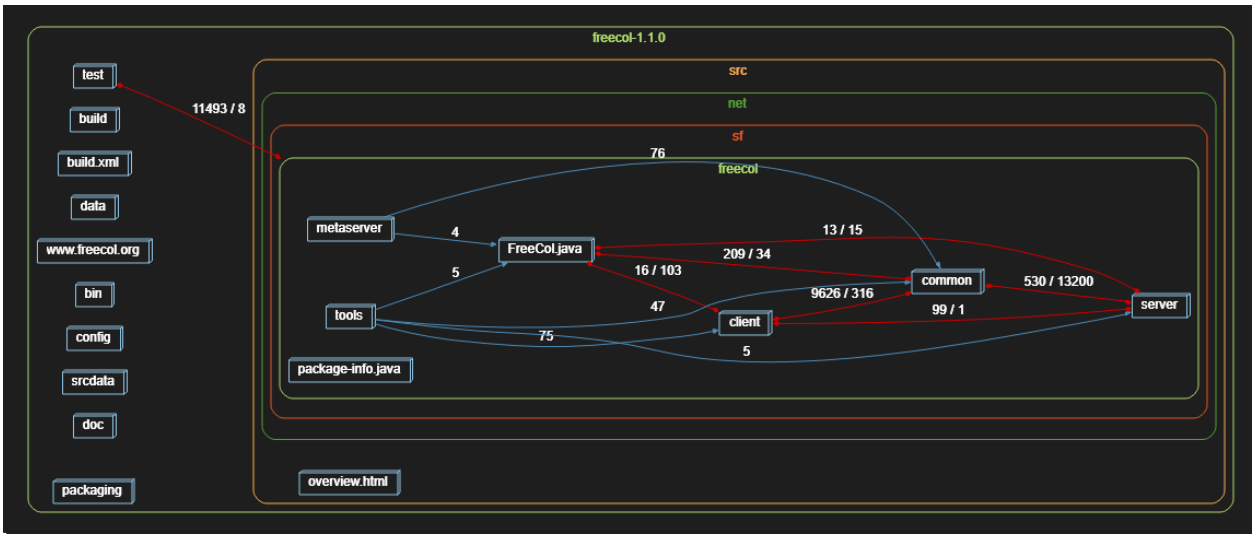
[1] https://scitools.com

[2] https://www.visual-paradigm.com/

11. ALLOCATION OF RESPONSIBILITIES WITHIN TEAM MEMBERS

Name of Team Member	Description of Responsibility	Allocation Unit
Ahmet Ensar Sönmez	*We did it all together.*	#
Eyüp Zafer Ünal	*We did it all together.*	#
Fevzi Kılas	*We did it all together.*	#

12. Appendix



Graph 1: dependency graph at product level