DAT321/DIT847:

Software Quality

Welcome to the examination for the *Software Quality* course. Each question has a number of points assigned shown in the square brackets. When the question is broken down into smaller sub-questions the part of the points for that specific sub-questions are also shown as following:

1. *[10 pts].*
    a. *[2 pts]*
    b. *[8 pts]*

The percentage of points and the corresponding grade is presented below (100 points in total):

% of points	DAT321	DIT847
[0, 50%)	U	U
[50%, 65%)	3	G
[65%, 85%)	4	G
[85%, 100%]	5	VG

Instructions about how to submit this exam are found in the Assignments section of Canvas (under the Written Exam Submission page). You must submit a PDF file with your answers. Make sure that you label your answers with the question numbers so we can identify which question your answer is referring to. You can choose any editor to write your answers.

Important: You must write clear, readable, understandable and unambiguous answers. An advice is to refer to the provided software product context used in this exam.

The questions in this exam refer to the **ISO 25010:2011** that categorises internal and external software quality attributes into eight characteristics.

Questions about the exam contact:
Francisco Gomes, tel. 031 772 69 51, francisco.gomes@cse.gu.se

The exam review will be done via Zoom scheduled for:
Date: 2020-11-20 between 09:00 – 10:30.
Zoom Link: https://chalmers.zoom.us/j/63042185246
Password: 646191

The questions in this exam are related to the context below. Note that you must justify your answers with i) the theory and terminology from software quality and ii) their connections to the elements in this context (e.g., teams, product, processes, architecture, customers, etc.).

You are hired by an online gaming company called OnGame-Co that creates one of the most popular **competitive online games** with millions of users. The software product for this exam is a real-time action game where several players (i.e., the end-users of the software) work together to beat the opposing team. Examples of similar games are League of Legends, Overwatch, Smite, etc.

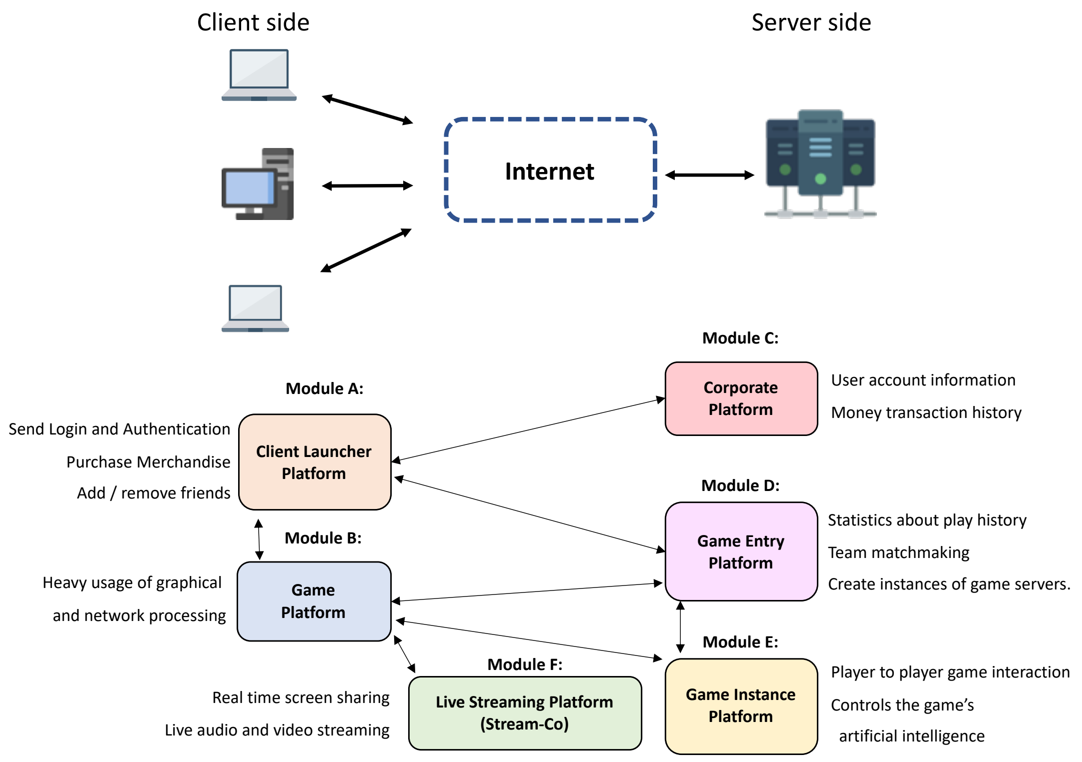
Features, business and customers:
In order to engage the community of players, OnGame-Co also developed a launcher platform where players first need to login to access the game. Before starting the actual game (i.e., game lobby), players can **hang out**, **chat** with other **online** players, **add friends** to play together, and even **buy merchandise** (t-shirts, accessories, toys) related to the game. To buy merchandise, players provide their credit card information to perform monetary transactions.

To ensure fairness in their matches, the game platform can create random teams of players with **similar gameplay statistics** (e.g., choice of characters, level of experience, user profile). This **matchmaking** feature is an **important feature** since different team composition is used by the Artificial Intelligence software of the game to adapt the game difficulty to match the player's experience level.

OnGame-Co also has a partnership with another company named Stream-Co that provides a **streaming platform** so that players can do live streaming of their gameplay. Examples of similar platforms are Twitch or Youtube. Note that **OnGame-Co does not develop the streaming platform**, but must **ensure** that its game platform can communicate with the platform developed by Stream-Co.

Design and architecture:
The software product uses a typical Client-server architecture (Figure 1), and players play live using an Internet connection. The game is composed of different **modules deployed in different parts of this architecture**. Each module's responsibilities are listed below; they are connected to the features of the system listed above. These modules exchange information and also depend on each other for the game to properly function.

Modules A and B are deployed in the **client** side of the architecture, whereas **Modules C, D and E** are deployed in the **server** side of the architecture. **Module F** is its own platform **developed and deployed by Stream-Co**. Our only responsibility to it is **to make sure the streaming interfaces can communicate with our gaming interface**s to ensure communication between platforms.


Client side
Server side
Internet
Module A:
Client Launcher Platform
Send Login and Authentication
Purchase Merchandise
Add / remove friends
Module B:
Game Platform
Heavy usage of graphical and network processing
Module C:
Corporate Platform
User account information
Money transaction history
Module D:
Game Entry Platform
Statistics about play history
Team matchmaking
Create instances of game servers.
Module E:
Game Instance Platform
Player to player game interaction
Controls the game's artificial intelligence
Module F:
Live Streaming Platform (Stream-Co)
Real time screen sharing
Live audio and video streaming

Development process and teams:
There are three teams in OnGame-Co. Team X is a team of 50 engineers composed into smaller teams of roughly equal size that work with the server side of the game (Modules C, D and E). Team Y is composed of 30 engineers working in the client side of the game evenly split between Modules A and B. Finally, Team Z is a small team of 4 engineers responsible for creating and maintaining the communication with Stream-Co (Module F).

The three teams are mainly composed of a mix of both junior and senior engineers (developers, testers, architects, managers, UX and game designers) that have used agile methods before. Each team has their own isolated development environment using version control. However, **only Team X is doing continuous integration**, such that teams Y and Z do not have the necessary tool chain (i.e., automation tools and servers) installed in their development environments.

Software Product – Future plans:
OnGame-Co is aiming to create a **new game** that runs on different hardware platforms including laptops, mobile phones and tablets. This new game is also independent from the existing game. Still, OnGame-Co aims to reuse Modules C and D of their current architecture to save time and costs with software development. Moreover, OnGame-Co aims to create two new teams:

- Team M: Responsible for developing the new client-side Module G for this new game;
- Team T: A small team composed of 5 test engineers whose sole responsibility is to plan, implement and manage all testing strategies for all games developed by OnGame-Co;

1. **[30 pts]** Using the context above, answer the following questions:

   a. [15 pts] Choose three software product quality's characteristics, **and, for each,** provide examples of why they are relevant to the software product at OnGame-Co. You can use the features listed in Figure 1 as sources for your examples.

Any three examples of the 8 below would work. The key point here is to connect the **description** of the software product quality characteristic (i.e., only Internal or External allowed, not quality in use) to the context above, specially the modules. The amount of points depends on the clarity of the justification behind the relevance. Ideally, for full points you should exemplify one Modules or Teams of the context provided.

(Each description of relevance: 3-4 points, Using Modules or Teams: 1-2 points).

(There should be 3 at least … if there are more, we will look at the totality of what you wrote. For instance, 4 examples in which only 3 of them are correct will not award you full points since there are still wrong descriptions provided.)

1. Security: This quality characteristic is relevant because users need to do login and have their own information stored in the program (money transactions, playing records, account information), which need to be accessible only by the corresponding player to avoid risks with stealing users' information. The key modules for this characteristic are Modules A and C.

2. Compatibility: This is relevant because the online game needs to share information with the streaming platform developed by Stream-Co, or with any banking app used to process users' purchases. Other examples would be the exchange of information from games installed in different clients, or other online store in case the merchandise is sold by other partners. So, your answer should include which modules should be communicating (e.g., Module F with B)

3. Portability: This is relevant for the development of the new game because the software needs to run on multiple hardware devices (tablets, laptops, mobile phones, etc.). This characteristic is particularly important for Team M and the development of the new game, so other examples related to the new module G or team M is also acceptable (if the examples are correct).

4. Functional suitability: This is relevant since the players have expectation on the features provided by both the Launcher and the game. Failing to deliver these features can cause players to stop playing the game or create unfair competitions (e.g., issues with matchmaking or the Artificial Intelligence of the software). Another example is the relevance of this characteristic for the streaming features of the game enabled by Module F. All modules here are seen as relevant.

5. Performance efficiency: This is relevant because the game runs on real-time. For instance, issues with high memory, processing or graphical usage or even low latency in the network (Module B) can ruin the gameplay experience. Another example on performance efficiency can be used for Module F where the resources must be sufficient to run both the game and the streaming platform in parallel.

6. Maintainability: This is relevant because the game needs to go through several updates to make sure we add new features (e.g., characters to the game or functionalities to the Launcher or the social platform) and those modifications cannot be disruptive of the functioning of the game. Another example is making sure Module F can be modified to cope with changes from Stream-Co, without breaking the game or the streaming feature.

7. Reliability: This is relevant because the game is competitive and runs live, so failures that lead to crashes of the game can ruin the gameplay experience or in serious e-sports competition this can cause severe damages in sponsoring or legal aspects. This is particularly important for modules in the Server side, since in case one client side, the architecture can still allow the game to continue, but crashing on the server-side means crashing for all players that are online. Important modules are Modules C, D, E and F.

8. Usability: This is relevant for players to be able to learn and keep playing the game and or the game launcher. Concrete examples should be connected to the game's interface, the matchmaking feature, the usability of the game launcher or streaming features within the game. Other examples are also connected to the development of the game's AI to provide fair gameplay experience. So, example of important modules are A, B, D and E.

b. [15 pts] Choose **one of the features** from the modules in Figure 1 and explain how we can evaluate **the quality of the chosen feature** from three different perspectives, namely, (i) an Internal Quality perspective, (ii) an External quality perspective, and (iii) a Quality in Use perspective.

Points are awarded using: 5- Clarity and quality of description of internal quality, 5- external quality and 5- quality in use. It is important to refer to all three perspectives of the same feature. A few examples:

Matchmaking: From (i) internal quality we can evaluate the quality of matchmaking by checking the coupling between modules so that the matchmaking feature can easily access data on player's record. Note that both are in the same module (Module D) to avoid unnecessary coupling between modules. For (ii) external quality, we can evaluate the quality of matchmaking by testing how the matchmaking works for players with different playing records that are depending on the outcomes of the game, so the matchmaking should be dynamic while the game is running. For (iii) quality in use we can evaluate the player's satisfaction for the various matchmaking done after a few games have been played; we can do this evaluation via a survey or asking the user how the quality of the match was in terms of difficulty.

Real-time screen sharing (Module F): (i) we can evaluate the internal design of this feature in relation to the interfaces implemented to enable communication between the game and the streaming platform, such that updates in screen sharing do not affect the game. (ii) for external quality, we can evaluate the impact of screen sharing on the game's performance such as latency or graphical performance while the game is running on different operating systems. (iii) for quality in use, we can evaluate this feature by seeing if users can effectively and efficiently enable or disable the screen sharing options live while playing a game online, or perhaps how the screen sharing is seen by the player by distinguishing chat messages from players in their team and messages from a larger audience of the player's streaming followers.

There are many other examples, but the key essence is to see the same features but based on different aspects, namely: (i) for internal quality they should be connected to artefacts, so the other modules, the architecture, code, design, etc; (ii) for external quality they should be connected to the game running which includes the resource capacity, live network communication, and other real-time aspects of the game; (iii) for quality in use they should be connected to the player's experience independently of the game's code or architectural designs.

2. **[15 pts]** OnGame-Co wants to foster sustainability in ICT. Using the Sustainability Awareness Framework, you must (i) choose or suggest **two features** to the game and explain the (ii) immediate, (iii) enabling and (iv) structural effects of those features. You must also indicate the **corresponding dimension** in which those effects occur.

   For this question you are free to choose to modify an existing feature (from Figure 1) or create a new feature. We refer to feature as a functionality in our software product. Make sure to clearly distinguish between the immediate, enabling and structural effects in your answer.

   Similar to question 1, there are many right answers here. Each example is 7.5 points. The key is to provide a clear connection between the immediate, enabling and structural effects (5-6 points for the quality of examples and connection), and the appropriate choice of feature and corresponding dimensions (1-2 points).

   Suggested feature: Monitor player interaction in chat (lobby and in game) to punish or bane players with toxic or discriminative behaviour. Immediate effect on individual dimension: Allow for more diverse and less stressful gaming experience; Enabling effect on social dimension: We will have fewer toxic players and create a safer community of players; Structural effect in the economic dimension: We may attract more players to spend their time and money in the game (e.g., a "negative economic effect" is also possible here where we have the toxic players leaving which we would lose some money).

   Suggested feature: Focus on retro-compatibility when implementing the new game for various hardware devices. Immediate effect on technical dimensional: Focus testing efforts to ensure portability to previous hardware devices, user interfaces and their operating systems; Enabling effect on economic / individual dimensional: Players don't need to buy new hardware (e.g., new phones, tablets or graphical cards) to keep playing the game throughout years; Structural effect on the environmental dimension: We reduce the amount of waste since players do not need to dispose of old hardware devices to keep playing the game.

Suggested feature: Create in-game currency that can be awarded based on loyalty and gameplay time. Immediate effect on technical dimensional: Update the design to handle (micro-)transactions when involving money or in-game currency; Enabling effect on economic dimension: Players don't need to spend a lot of money to unlock or purchase various merchandise inside the game; Structural effect on the individual dimension: Players will spend more time playing the game and less time outside engaging with players outside the virtual world.

Suggested feature: Allow the graphical user interface to be more accessible via, e.g., options for colour blind friendly interface colour palettes – or improve textual interface to follow dyslexia-friendly guidelines. Immediate effect on the technical level: Improve the accessibility of the system for various types of user; Enabling effect on the individual level: Attract players with disabilities and creating a more inclusive software. Structural effect on the social level: Strengthen inclusion and diversity in the online gaming community.

3. **[30 pts]** Considering the decision to migrate all teams to Continuous Integration and the specific composition of Team T (dedication to testing), answer the following questions:

   a. [10 pts] Choose two different levels of testing. Considering any of the features in Figure 1, provide at least two examples of test cases for each level of testing chosen.

      This means that each test case description accounts for 2.5 points (two test cases for each of the two levels of testing – 5 examples in total). The levels of testing are: unit, integration, system or acceptance.

      For each test case, you get points if you include (i) correct reasoning that the test case verifies the feature at the right level of testing (1.5 points), and (ii) comprehensible description of the test case as a scenario with conditions and expected output (1 point).

      A few examples below:

      Chosen feature: Login to Game Launcher
      For Unit-level: an example of a test case is to verify the code class that checks whether a valid username and password provided matches the user information stored in the database. **Integration-level:** a test case is to verify whether the user authentication information sent by Module A can be properly received and handled by Module C which then returns the result of the authentication back to Module A. **System-level:** a test case is to verify whether players can login at the same time in multiple sessions (i.e., opening several launchers in different computers); or if we talk about the new game is to verify whether players can properly login from different devices; **Acceptance-level:** a test case is to verify whether players have access

to the right game features after a successful login. Alternatively, we can also verify whether players with invalid login are indeed not able to access any features from the launcher.

Chosen feature: Player to player game interaction
Unit-level: a test case to verify whether the actions of a game's character (e.g., heroes in the game) implemented as, e.g., functions or events in the code work as expected. **Integration level:** a test case to verify whether the actions sent by the player's client side (Module B) are indeed received and processed by Module E that then can provide feedback to the player (Module B). **System-level:** a test case to verify whether several players (e.g., artificial players, or using specific hardware configuration on the client side) in their respective client side can interact with each other using their internet connection to the server. For system-level, a full game is not needed, rather we can have fixed definitions of the game. **Acceptance-level:** a test case to verify whether actual players (human testers) in different clients (perhaps distributed geographically) can start and finish a game match using their internet connection and specific setup.

b. [10 pts] OnGame-Co decided to migrate all its teams to develop using Continuous Integration. List **two activities** you would do or teach to the teams before introducing CI. Justify why those activities would be relevant to introduce CI in OnGame-Co

Here you would receive 5 points for the clarity and correctness of your activity. Correctness is analyzed if the activity is related to the main requirements of adopting CI: (i) adopting and using configuration management, (ii) developing team practices for continuously testing, and (ii) making small but frequent changes in the software product, (iii) using version control software (git, svn, etc.), (iv) instrumenting automated builds and the entire tool chain necessary to continuously and autonomously build and test the software after modifications.

These activities must be well described (2 points) and properly connected to the context (3 points). Example of answers are:

The context states that not all teams have the instrumentation for CI, besides version control, so we need to train the engineers in all teams to use automation servers such as Jenkins or Travis CI to connect with their code repositories.

The context states that there will be one team dedicated for testing, so we need to train this team of testers in enabling automated testing in the CI toolchain.

The context states that teams are familiar with agile, so they can handle more frequent changes, but there is not information on their capability of continuously testing, so we can offer training in properly training them in using test-driven development, or the discipline to test more often.

The context states that only one team uses CI, so we need to develop a strategy for having integration at different stages and distinguish, e.g., development branches, integration branches and production branches along with corresponding test suites to allow these three different pipelines to revert modifications that break the system, without risking break the game that is already released to production.

c. [10 pts] Introducing CI will affect all teams in OnGame-Co. Particularly for the role of Team T in the development process, list one advantage and one disadvantage that CI will bring. Note that Team T is a team dedicated to testing, so your advantage and disadvantage **must** be connected to software testing.

Each answer is worth 5 point, where 3 points is the correctness of the (dis)advantage in relation to the context, and 2 points is its explicit connection to testing. You can connect to testing by relating to concepts in testing, such as failure, faults, errors, levels of testing, quality of tests, testing techniques, etc.

Examples of disadvantages with testing and CI:
- Testing is that testing quickly becomes costly, since the number of tests grows quickly with the number of changes introduced, such that releasing updates or new versions of the game can take longer. So, we need proper testing strategies like test prioritization or selection.
- The automated testing becomes dependent from the toolchain, such that failures in the toolchain are disruptive and do not necessarily mean that the game itself has a fault.
- Team T needs to test the integration with the streaming platform which is not developed by OnGame-Co, so the CI can test Module F in connection to our own modules, but not in connection to the releases of our partner's streaming software (Stream-Co), since they may or may not have their own CI processes for the streaming platform.

Examples of advantages with testing and CI:
- Teams will be able to have parallel development of their modules and can automatically verify the integration of all their modules. This allows to identify which modules are causing failures in the build, which can improve maintainability of the game.
- Each team can focus development and testing on their own modules, whereas Team T can focus on testing the integration of all modules using the CI instrumentation.
- Team T can create different testing strategies to separate functional testing and non-functional testing (e.g.., memory and network performance of the game) as separate pipelines in the CI.

4. **[15 pts]** Considering internal quality measures and quality attributes, answer the following questions:

a. [10 pts] An analysis of the technical debt revealed that all modules need refactoring and have the same debt, such that restoring the maintainability of each module would take

10 days. You only have resources to choose three modules to reduce this technical debt. **Which modules** would you choose and **why**?

Two required Modules in your answer is Module C and D (2 points), because the context states that they will be reused in the development of the new game (3 points). Therefore, the technical debt that they currently hold would already impact the new game to be developed that would start with technical debt already (2 points).

The third module will vary depending on your answer (3 points). A few examples: Any of the server-side modules since they can affect all players if there are any delays in the maintenance of the software; the modules on the client-side can be justified since they enable the player's access to the game, or attract more players, so design issues there are more apparent to the user. Lastly, Module F could be justified due to its dependency to the streaming platform which needs to be easy to maintain (i.e., make updates) in case there are relevant changes introduced by Stream-Co. Other acceptable justifications are connected to the size of the team of the corresponding module, such that larger teams can spend more time fixing technical debt than smaller teams (since the context explains that the teams have roughly the same skills and experience).

b. [5 pts] You were asked to design a two-dimensional chart (x-axis and y-axis) to visualise the internal quality of the different modules in Figure 1. Which two quality measures would you show in this chart? Which quality attributes would be easier to understand using your proposed visualisation?

Note that the question says Internal Quality, so metrics derived from software execution are not applicable (test results, network latency, resource usage, number of players joining or leaving). The metrics must be connected to the software artefacts. Given the focus on architecture, one of the axes (e.g., y-axis) should include Henry & Kafura complexity, Fan-in or fan-out, number of method calls to other modules (choice or right measure = 1 point), or other measure that makes it easier to understand modularity or coupling (2 points), since the each team is responsible for a module, so this would allow teams to have a shared view of the module dependencies (1 point). The second axis can be a complementary metric with many options. The important part is providing the right justification for the combination chosen. For instance (2 points):

- number of changes or commits, allows us to see which modules change more often;
- size of the module in lines of code, allows us to see which modules are bigger or smaller to allocate refactoring resources;
- cyclomatic complexity, allows us to see which modules are easier or harder to test;
- number of faults (not failures!), allows us to see which modules often require more debugging effort

5. **[10 pts]** Every workday, the Continuous Integration pipeline merges all code commits and automatically builds a version of the game. Those commits vary in size (Lines of Code – LOC) and in which module was modified.

In order to properly plan for the upcoming weeks of development, OnGame-Co wants to predict whether a given build of your CI will fail or not based on the size of the commit made and which module was modified. You are able to collect data from the past six months of modifications to the modules (example of the dataset shown below):

Build status	**Size of Commit (LOC)**	**Module**
Successful	120	Module A
Failed	150	Module B
Failed	200	Module A
Successful	85	Module C
Successful	50	Module F
Failed	350	Module D
…		

Write down the *mathematical model definition* for this prediction using *any* variable names and priors of your choice. State the ontological and epistemological reasons for your likelihood. Remember to clearly state and justify the choices and assumptions regarding your model.

In term of the reasoning, one should argue for a Binomial(n,p) where N is the size of the commit in LOC and p is the probability of a build failing (3 points). Alternatively, if you justify that the build status is in relation to a **single** line of code of changed (N = 1, "one trial") that would imply a Bernoulli(p) likelihood (1 point only if it was a Bernouli, since its not practical to analyse in terms of single LOC changes).

The second predictor would be the changed Module but to account for the varying effect of each module we would need a varying intercept (3 points – 1 point if it was not with varying effects).

So, with math notation, the model would be (4-5 points in total for the right math notation):

(1 points) Fail_i~ Binomial(1,p_i) or Bernoulli(p_i)
(1 points) p_i = α + β_loc × LOC_i + α_MODULE[i]
(1 point) α ~ Normal(0,10)
(1 point) β_loc ~ Normal(0,2.5)
(1 point) a_MODULE[i] ~ Exponential(1) (or HalfCauchy/LogNormal/Weibull/whatever)

You can get 1-2 points depending on the clarity and quality of your justification of using wide priors.