A.Y. 2023-2024 Software Engineering 2
Requirement Engineering and Design Project: goal, schedule, and rules

READ THIS VERY CAREFULLY
NO EXCUSE FOR IGNORING WHAT WE WRITE HERE

Table of contents

1 Goal and approach

The objective of this project is to apply in practice what you learn during lectures with the purpose of becoming familiar with software engineering practices and able to address new software engineering issues in a rigorous way. The project includes two assignments:

1. The preparation of a Requirement Analysis and Specification Document (RASD) for a problem we provide you.
2. The definition of the Design Document (DD) for the system considered in point 1 above.

The two assignments will be reviewed during the final discussions that will take place during the winter exam sessions according to a schedule that will be proposed in the forthcoming months. The evaluation will assess the quality of the artifacts you prepare (accurateness, completeness, soundness) and the quality of your presentation (if you are able to explain your point in an appropriate way and if your presentation fits in the allowed time). Please check the introduction to the course for more information on the evaluation criteria for the R&DD project. The two assignments are described in the rest of this document.

2 Project schedule

- Group registration deadline: 22/10/2023
- RASD submission deadline: 22/12/2023
- DD submission deadline: 07/01/2024
- Final presentation: to be scheduled

All deadlines are assumed to expire at **23:59** (Rome Time).
Note: You can submit before the deadlines, if you want/need.

3 Rules

- This assignment is optional and replaces part of the written exam of the Software Engineering 2 course.
- The project is developed in groups of two or three persons. Groups composed of a single student are allowed even if strongly discouraged. The assignments and the corresponding expectations of the professor will be calibrated based on the size of the group.
- Each group interested in taking the project must register itself following the steps listed in Section 4. “Mixed” groups involving students of the three sections are allowed. When registering, these groups must indicate a single “reference” professor. This will be the one holding the discussion at the end of the course and deciding the grade. The choice is up to the students, but if we will realize that there is an unbalance among the groups under the responsibility of each of us, we may change your reference professor. In this case, we will inform you a few days after the group registration deadline.
- Each group MUST provide the requested artifacts within the stated deadlines. A delay of a few days, if notified in advance to the reference professor, will be tolerated, but it will also result in a penalty in the final score. These artifacts will be presented to the reference professor in a final meeting that will be scheduled later.
- Each group MUST release artifacts by committing and then pushing into the main branch of the git repository created for the project (see the following section).
- Each group MUST use the repository not only to push the final versions of deliverables, but also for intermediate versions. We want to see commits performed by all group members. In the case a group wants to use collaborative writing tools (e.g., Google Docs) that keep track of individual contributions, the group will include in the readme file associated with the git repository the link to the online document, making sure that through that link the reference professor will be able to inspect the contributions to the document.
- The material included in your artifacts is not fixed in stone. You can (and are encouraged to) provide updates at any point before the final submission deadline if you think these are needed.
- During the development of the project each group will keep track of the number of hours each group member works toward the fulfillment of each deadline.
- For any question related to the project that could be interesting also for the other groups, please use the forum available on the Webeep website. We will answer as soon as possible.

4 Group registration and organization of your repository

Students autonomously form groups and register by following these steps:

1. Create a private repository for your project on GitHub (https://github.com). Please note that, as students, you have the possibility to create a private GitHub repository for free. Your repository should be named by combining the names of all group members. For instance, BianchiRossiVerdi will be the name of the repository of the group composed of the students Tommaso Bianchi, Maria Rossi e Veronica Verdi. Make sure that all group members have a Github account and have access to the repository (write permission). Moreover, invite your reference professor (GitHub accounts *matteocamilli* for Prof. Camilli, *dinitto* for Prof. Di Nitto, and *matteo-g-rossi* for Prof. Rossi) to access your repository (read permission is sufficient). Note: in past years we noticed that GitHub has set some restrictions concerning the creation of repositories from

accounts of people from specific countries. If you are experiencing this problem, you can use Bitbucket instead (https://bitbucket.org/product/). We do not suggest this one for all groups because it has other limitations concerning the maximum number of private repositories you can have.

2. Register your group by filling in the following form https://forms.office.com/e/MsXnwyAWbe. Do not forget to include in the form all relevant data.
3. Create a directory for each of the documents you will be working on.
4. Moreover, create a directory called *DeliveryFolder* where, by the due deadlines, you will commit and push the pdf version of your documents (name it RASDv1.pdf or DDv1.pdf, depending on the document you are releasing) plus any additional file you may want to include (e.g., the Alloy model and/or any UML model).
5. After the submission deadline, should you need to update your document, you can commit in the same folder another pdf file with an increased version number, e.g., RASDv2.pdf. The new file should include a section that describes the performed changes.

5 The problem: CodeKataBattle

CodeKataBattle (CKB) is a new platform that helps students improve their software development skills by training with peers on *code kata*[1] . Educators use the platform to challenge students by creating code kata battles in which teams of students can compete against each other, thus proving (and improving) their skills.

A code kata battle is essentially a programming exercise in a programming language of choice (e.g., Java, Python). The exercise includes a brief textual description and a software project with build automation scripts (e.g., a Gradle project in case of Java sources) that contains a set of test cases that the program must pass, but without the program implementation. Students are asked to complete the project with their code. In particular, groups of students participating in a battle are expected to follow a *test-first* approach[2] and develop a solution that passes the required tests. Groups deliver their solution to the platform (by the end of the battle). At the end of the battle, the platform assigns scores to groups to create a competition rank.

Each battle created by an educator belongs to a specific tournament. Tournaments are created by an educator who can then grant to other colleagues the permission to create battles within the context of a specific tournament. When a new tournament is created, all students subscribed to the CKB platform are notified and they can subscribe by a given deadline. If they subscribe, they are notified of all upcoming battles created within that tournament. To create a new battle, an educator uses the CKB platform to perform the following steps:

- upload the code kata (description and software project, including test cases and build automation scripts),
- set minimum and maximum number of students per group,
- set a registration deadline,
- set a final submission deadline,

[1] A kata is an exercise in karate where you repeat a form many, many times, making little improvements in each. Code Kata is an attempt to bring this element of practice to software development. See more at http://codekata.com/

[2] More at https://en.wikipedia.org/wiki/Test-driven_development

- set additional configurations for scoring (see further details in the following).

After creation of a battle, students use the platform to form teams for that battle. In particular, each student can join a battle on his/her own or by inviting other students (respecting the minimum and maximum number of students per group set for that battle). When the registration deadline expires, the platform creates a GitHub repository containing the code kata and then sends the link to all students who are members of subscribed teams. At this point, students can start working on the project. Students are asked to fork the GitHub repository of the code kata and set up an automated workflow through GitHub Actions that informs the CKB platform (through proper API calls) as soon as students push a new commit into the main branch of their repository. Thus, each push before the deadline triggers the CKB platform, which pulls the latest sources, analyzes them, and runs the tests on the corresponding executables to calculate and update the battle score of the team.

The score is a natural number between 0 and 100 determined by considering some mandatory factors evaluated in a fully automated way, and optional factors evaluated manually by educators.
Mandatory automated evaluation includes:

- functional aspects, measured in terms of number of test cases that pass out of all test cases (the higher the better);
- timeliness, measured in terms of time passed between the registration deadline and the last commit (the lower the better);
- quality level of the sources, extracted through static analysis tools that consider multiple aspects such as security, reliability, and maintainability (the higher the better). Aspects are selected by the educator at battle creation time.

Optional manual evaluation includes:

- personal score assigned by the educator, who checks and evaluates the work done by students (the higher the better).

The CKB platform automatically updates the battle score of a team as soon as new push actions on GitHub are performed. So, both students and educators involved in the battle can see the current rank evolving during the battle. When the submission deadline expires, there is a consolidation stage in which, if manual evaluation is required, the educator uses the CKB platform to go through the sources produced by each team to assign his/her score. Once the consolidation stage finishes, all students participating in the battle are notified when the final battle rank becomes available.

At the end of each battle, the platform updates the personal tournament score of each student, that is, the sum of all battle scores received in that tournament. Thus, for each tournament, there is a rank that measures how a student's performance compares to other students in the context of that tournament. This information is available for all students and educators subscribed to the CKB platform, that is, all users can see the list of ongoing tournaments as well as the corresponding tournament rank.

When an educator closes a tournament, as soon as the final tournament rank becomes available the CKB platform notifies all students involved in the tournament.

The CKB platform also includes gamification badges: elements in the form of rewards that represent the achievements of individual students. Badges are defined by educators when they create a tournament. Each badge has a title (e.g., “top committer”) and one or more rules (i.e., simple Boolean

properties) that must be fulfilled to achieve the badge. Thus, each badge is assigned to one or more students depending on the rules checked at the end of the tournament. The following are examples of possible badges:

- Title: tournament participant
  - Rules: { *tot_attended_battles > 0* }
- Title: top committer
  - Rules: { *tot_commits_student* == *max_tot_commits* }

Where *tot_attended_battles*, *tot_commits_student* and *max_tot_commits* are pre-defined variables that represent, respectively, the total number of battles the student has been involved in, the number of commits carried out by the student and the maximum total number of commits considering all students.

As mentioned above, educators can create new badges and define new rules as well as new variables associated with them. Variables can represent any piece of information available in CKB relevant for scoring. It is up to you to identify such information and a simple language for creating new variables, rules, and badges.

Badges can be visualized by all users. In particular, both students and educators can see collected badges when they visualize the profile of a student.

6 Project Scope

You have been contacted to define the RASD and DD for CodeKataBattle. The task is different for groups of different sizes. More specifically:

- **Groups composed of a single student** will focus on all aspects of CodeKataBattle **excluding**:
  - the integration with the tools performing static analysis on the code and contributing to the computation of team scores;
  - the gamification aspects.
- **Groups composed of two students** focus on all aspects of CodeKataBattle **including** the integration with the tools performing static analysis on the code and contributing to the computation of team scores, and **excluding** the gamification aspects.
- **Groups of three students** will focus on all aspects of of CodeKataBattle **including**:
  - the integration with the tools performing static analysis on the code and contributing to the computation of team scores;
  - the gamification aspects.

Examples of similar (but different) platforms

Codewars https://www.codewars.com/

7 The documents to be created

Each document you produce will include the following elements:

- **A FRONT PAGE** that includes the project title, the version of the document, your names and the release date.
- **A TABLE OF CONTENTS** that includes the headers of the first three levels of headings in your document, with the corresponding page number. At the beginning of this document, you find a table of contents that you can use as an example. Since in this document there are no level three headings (e.g., 3.1.1), they are not part of the table of contents.

The specific characteristics of each document are described in the next subsections.

7.1 Assignment 1 - RASD

The *Requirements analysis and specification document (RASD)* contains the description of the scenarios, the use cases that describe them, and the models describing the requirements and specification for the problem under consideration. You are to use a suitable mix of natural language, UML, and Alloy. Any Alloy model should be validated through the tool, by reporting the models obtained by using it and/or by showing the results of assertion checks. Of course, the initial written problem statement we provide suffers from the typical drawbacks of natural language descriptions: it is informal, incomplete, uses different terms for the same concepts, and the like. You may choose to solve the incompleteness and ambiguity as you wish but be careful to clearly document the choices you make and the corresponding rationale. You will also include in the document information on the number of hours each group member has worked towards the fulfillment of this deadline. As a reference structure for your document, you should refer to the one reported below that is derived from the one suggested by IEEE. Please include in the document information about the effort spent by each group member for completing this document.

1. **INTRODUCTION**
   - A. *Purpose*: here we include the goals of the project
   - B. *Scope*: here we include an analysis of the world and of the shared phenomena
   - C. *Definitions, Acronyms, Abbreviations*
   - D. *Revision history*
   - E. *Reference Documents*
   - F. *Document Structure*
2. **OVERALL DESCRIPTION**
   - A. *Product perspective*: here we include scenarios and further details on the shared phenomena and a domain model (class diagrams and state diagrams)
   - B. *Product functions*: here we include the most important requirements
   - C. *User characteristics*: here we include anything that is relevant to clarify their needs
   - D. *Assumptions, dependencies and constraints*: here we include domain assumptions
3. **SPECIFIC REQUIREMENTS:** Here we include more details on all aspects in Section 2 if they can be useful for the development team.
   - A. *External Interface Requirements*
     - A.1 *User Interfaces*
     - A.2 *Hardware Interfaces*
     - A.3 *Software Interfaces*

A.4 *Communication Interfaces*

B. *Functional Requirements*: Definition of use case diagrams, use cases and associated sequence/activity diagrams, and mapping on requirements

C. *Performance Requirements*

D. *Design Constraints*

D.1 *Standards compliance*

D.2 *Hardware limitations*

D.3 *Any other constraint*

E. *Software System Attributes*

E.1 *Reliability*

E.2 *Availability*

E.3 *Security*

E.4 *Maintainability*

E.5 *Portability*

4. **FORMAL ANALYSIS USING ALLOY:** This section should include a brief presentation of the main objectives driving the formal modeling activity, as well as a description of the model itself, what can be proved with it, and why what is proved is important given the problem at hand. To show the soundness and correctness of the model, this section can show some worlds obtained by running it, and/or the results of the checks performed on meaningful assertions.
5. **EFFORT SPENT:** In this section you will include information about the number of hours each group member has worked for this document.
6. **REFERENCES**

7.2 Assignment 2 - DD

The *Design document (DD)* must contain a functional description of the system, and any other view you find useful to provide. You should use all the UML diagrams you need to provide a full description of the system. Alloy may also be useful, but not mandatory. You will also include information on the number of hours each group member has worked towards the fulfillment of this deadline. As a reference structure for your document please refer to the following one:

1. **INTRODUCTION**
   - A. *Purpose*
   - B. *Scope*
   - C. *Definitions, Acronyms, Abbreviations*
   - D. *Revision history*
   - E. *Reference Documents*
   - F. *Document Structure*
2. **ARCHITECTURAL DESIGN**
   - A. *Overview*: High-level components and their interaction
   - B. *Component view*
   - C. *Deployment view*
   - D. *Runtime view*: You can use sequence diagrams to describe the way components interact to accomplish specific tasks typically related to your use cases

E. *Component interfaces*
F. *Selected architectural styles and patterns*: Please explain which styles/patterns you used, why, and how
G. *Other design deci*sions

3. **USER INTERFACE DESIGN**: Provide an overview on how the user interface(s) of your system will look like; if you have included this part in the RASD, you can simply refer to what you have already done, possibly providing here some extensions if applicable.
4. **REQUIREMENTS TRACEABILITY**: Explain how the requirements you have defined in the RASD map to the design elements that you have defined in this document.
5. **IMPLEMENTATION, INTEGRATION AND TEST PLAN**: Identify here the order in which you plan to implement the subcomponents of your system and the order in which you plan to integrate such subcomponents and test the integration.
6. **EFFORT SPENT:** In this section you will include information about the number of hours each group member has worked for this document.
7. **REFERENCES**