Politecnico di Milano
Computer Science and Engineering
Software Engineering II

# RASD - CodeKataBlade

José Alejandro Sarmiento

December 11, 2023

v1.0

# Contents

# 1 INTRODUCTION

## 1.1 Purpose

The primary purpose of the CodeKataBattle (CKB) platform is to provide an environment for students to enhance their software development skills through collaborative learning and friendly competition and for educators to set up these scenarios. The platform facilitates this by allowing educators to setup tournaments where code kata battles are organized. Here students form teams and showcase their coding abilities in a test-driven development (TDD) framework.

**A) Skill Enhancement through Practice:**

CKB serves as a virtual arena where students can test and improve their programming abilities by actively participating in code kata battles.

**B) Educator-Guided Learning:**

The platform must allow for educators to create tournaments, compose battles and perform manual evaluations on top of automated ones. This ensures that the learning experience aligns with the curriculum and instructional goals.

**C) Automated Evaluation:**

CKB should have an automated evaluation system that provides feedback to students based on objective criteria. Specifically, the criteria the evaluation is based upon is the number of passed test cases set up by the battle organizer and the timeliness.

**D) Competition and Recognition:**

The platform should possess a leaderboard functionality that allows participants to gauge their performance and see how their skills compare to other participants.

In summary, CodeKataBattle aims to create a learning platform, combining hands-on coding practice, collaborative teamwork and automated feedback, all under the guidance of educators.

Knowing this, the S2B will have a certain set of goals to achieve:

- **G1:** Every educator should be able to create a tournament.
- **G2:** When crearing a tournament, the educator doing so should be able to set a registration deadline.
- **G3:** Every educator that owns a tournament should be able to invite other educators to it.

- **G4:** Every educator that belongs to a tournament should be able to create battles.
- **G5:** When creating a battle, the educator doing so should be able to set the programming language, a decription of the problem, the test cases which will evaluate the students code, the build automation scripts, the registration deadline, the final submission deadline, the minimum and maximum number of participants per group.
- **G6:** Every student should be able to see the list of tournaments and register to them before the registration deadline.
- **G7:** Every student should be able to see the description of the battles of a tournament they belong to and register to them before the registration deadline by themselves or with other students by inviting them to join or by accepting another student's invitation.
- **G8:** Every time a student makes a submission to a battle, the platform should evaluate it and update the leaderboard of the battle accordingly.
- **G9:** The evaluation carried on by the platform should be performed with the build automation scripts set by the educator and should be based on the test cases set by the educator and the timeliness of the submission.
- **G10:** Every student and educator should be able to see the leaderboard of a battle they belong to.
- **G11:** Once the battle ends, the educator that created it should be able to manually evaluate the code of the students if they so desire to and set a grade for each student.
- **G12:** Once the educator consolidates the results of a battle, the students that participated in it should be notified, the final results of the battle should be displayed to everyone. The leaderboard of the tournament the battle belongs to should be updated by adding for each student the battle score to the sum of all the other battles they have participated on.
- **G13:** Every student and educator should be able to see the leaderboard of every tournament.
- **G14:** An instructor that owns a tournament should be able to close it.
- **G15:** Once the owner of a tournament closes it, the platform should notify all the students once the leaderboard of the tournament is available.
- **G16:** When a tournament is created, all students should be notified.
- **G17:** When a battle is created, all students participating in the tournament the battle belongs to should be notified.

## 1.2 Scope

The education sector in software development is an ever-growing field given the nature of the topic being teached.

The CodeKataBattle (CKB) platform aims to provide a competitive twist to the learning by facilitating the creation of competitions in the form of tournaments where students can showcase their abilities.

These are managed by educators who can create the aforementioned tournaments and compose battles where they can decide the programming language, the description of the problem and the test cases which will evaluate the students code among other things.

This way, through the use of GitHub, the platform can automatically evaluate the code of a team.

In the environment of the CodeKataBattle system the following actors are identified:

**Students:** These are the primary users of the CKB system. They participate in battles and tournaments to improve their software development skills. They can form teams, work on code katas, and submit their solutions.

**Educators:**: They are responsible for creating and managing battles and tournaments. They set various parameters for battles, evaluate the solutions submitted by students, and provide feedback.

**GitHub:** The CKB system integrates with GitHub for managing code kata projects and tracking students' commits. GitHub is a widely used platform for version control and collaboration in software development.

Having all of this in mind, the following are the World and Shared phenomena that the system will have to deal with:

**World Phenomena:**

- **W1:** A group of students organizes themselves to participate in a battle together.
- **W2:** A group of students registered to a battle fork the repository of the battle.
- **W3:** Students set up an automated workflow through GitHub Actions that will notify the system every time a commit is pushed to the main branch of the forked repository of a battle.
- **W4:** A student pushes a commit to the main branche of a forked repository of a battle.
- **W5:** An educator possesses a specific evaluation criteria through which they perform their manual evaluations.

**Shared Phenomena:**

Controlled by the world and observed by the system.

- **SP1:** An educator creates a tournament with a set of attributes.
- **SP2:** An educator creates a battle with a set of attributes inside a tournament.
- **SP3:** An educator invites other educators to a tournament.
- **SP4:** A student registers to a tournament.
- **SP5:** A student or a team of students registers to a battle.
- **SP6:** GitHub notfies the system that a student has pushed a commit to the main branch of a forked repository of a battle.
- **SP7:** An educator performs manual evaluations on the submissions of a battle they own.
- **SP8:** An educator consolidates the results of a battle.
- **SP9:** An educator closes a tournament.

Controlled by the system and observed by the world.

- **SP10:** The system notifies all the students that a tournament has been created.
- **SP11:** The system notifies all the students participating in a tournament that a battle has been created.
- **SP12:** The system displays the leaderboard of tournament while it is active and after it has ended.
- **SP13:** The system displays the leaderboard of a battle while it is active and after it has ended.
- **SP14:** The system updates a battle's leaderboard every time a new evaluation is performed.
- **SP15:** The system notifies all the students participating in a battle that the final results have been consolidated.
- **SP16:** The system updates the leaderboard of a tournament every time a battle ends.
- **SP17:** The system notifies all the students participating in a tournament that it has ended.

## 1.3 Definitions, Acronyms, Abbreviations

### 1.3.1 Definitions

- **Tournament:** A tournament is a space where educators can create battles and students can register to them. It has a registration deadline and a leaderboard that is updated every time a battle ends.
- **Battle:** A battle is a space where students can register to and submit their solutions to a problem. It has a registration deadline, a final submission deadline, a leaderboard that is updated every time a new evaluation is performed and a set of test cases that will be used to evaluate the submissions.
- **GitHub:** GitHub is a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere.
- **GitHub Actions:** GitHub Actions is a CI/CD tool that allows you to automate your software development workflows in the same place you store code and collaborate on pull requests and issues.
- **Educator:** An educator, in the context of the system, is a user of the platform that can create tournaments and battles, invite other educators to tournaments, perform manual evaluations on the submissions of a battle they own and consolidate the results of a battle.
- **Student:** A student, in the context of the system, is a user of the platform that can register to tournaments and battles, create teams and submit their solutions to a battle.
- **Build Automation Scripts:** Build automation scripts are scripts that are run automatically by the system every time a commit is pushed to the main branch of the forked repository of a battle. They are used to evaluate the submissions of the students.
- **Test Case:** A test case is a set of conditions under which a tester will determine whether an application, software system or one of its features is working as it was originally established for it to do.
- **Timeliness:** Timeliness is the quality of doing something or producing something at the right time. In the context of the system, it refers to the time in which a student submits their solution to a battle with respect to the start of the battle and the final submission deadline.

### 1.3.2 Acronyms

- **CKB:** CodeKataBattle
- **S2B:** System to Be
- **TDD:** Test-Driven Development

- **CI/CD:** Continuous Integration/Continuous Delivery
- **UI:** User Interface
- **API:** Application Programming Interface
- **UML:** Unified Modeling Language

#### 1.3.3 Abbreviations

- **Gn:** Goal number n
- **Wn:** World Phenomena number n
- **SPn:** Shared Phenomena number n
- **Dn:** Domain Assumption number n
- **Rn:** Requirement number
- **UCn:** Use Case number n

## 1.4 Revision history

## 1.5 Reference Documents

The specification of the RASD and DD assignment of the Software Engineering II course, held by professor Matteo Rossi, Elisabetta Di Nitto and Matteo Camilli at the Politecnico di Milano, A.Y 2023/2024;

## 1.6 Document Structure

- **1. Introduction:** This section provides an overview of the entire document. It identifies the product's goals and the scope of the system along with the definitions, acronyms, abbreviations and the revision history.
- **2. Overall Description:** This section provides a general description of the product and its functionality. It also describes the user characteristics, assumptions, dependencies and constraints.
- **3. Specific Requirements:** This section contains all the software requirements to a level of detail sufficient to enable designers to design a system to satisfy those requirements, and testers to test that the system satisfies those requirements.
- **4. Formal Analysis Using Alloy:** This section contains a formal analysis of the system using the Alloy language.
- **5. Effort Spent:** This section contains the number of hours each group member has worked on the document.
- **6. References:** This section contains all the reference documents used to create this document.

# 2 OVERALL DESCRIPTION

## 2.1 Product perspective

### 2.1.1 Scenarios

- **1. Educator creates a tournament:**

  An educator called Adam wants to create a tournament for his students. He goes to the CKB platform and creates a tournament with a registration deadline of 2 weeks from now. The platform notifies all the students that a tournament has been created. He then invites his colleagues to the tournament so when the tournament starts they can create battles.

  **Goals:** G1, G2, G3, G16. **Phenomena:** SP1, SP3, SP10.

- **2. Student registers to a tournament:**

  A student called Bob receives a notification that a tournament has been created by educator Adam. He goes to the CKB platform and registers to the tournament before the registration deadline.

  **Goals:** G6, G16. **Phenomena:** SP4, SP10.

- **3. Educator creates a battle:**

  Educator Charlie wants to create a battle for the tournament he was invited to by educator Adam. He goes to the CKB platform and creates a battle with a registration deadline of 1 week from now, a final submission deadline of 2 weeks from now, a minimum of 1 participant per group and a maximum of 3 participants per group. He also sets the programming language to Java, the description of the problem and uploads the build automation scripts and test cases which will evaluate the students code. The platform notifies all the students participating in the tournament that a battle has been created.

  **Goals:** G4, G5, G17. **Phenomena:** SP2, SP11.

- **4. Students registers to a battle:**

  Bob receives a notification that a battle has been created by educator Charlie. He goes to the CKB platform and reads the description of the battle. He then contacts his friends Danielle and Eve, who are also inside educator Adam's tournament, and they decide to form a team and register to the battle together. Charlie invites Danielle and Eve to the battle and they accept.

  **Goals:** G7, G17. **Phenomena:** W1, SP5, SP11.

- **5. Student submits a solution to a battle:**

  After the registrarion deadline ends, the system sends the link to the newly created GitHub repository to the students that registered to the battle. Bob, Danielle and Eve fork the repository and set up an automated workflow through GitHub Actions that will notify the system every time a commit is pushed to the main branch of the forked repository of the battle. After that, they start working on the problem and push a commit to the main branch of the forked repository of the battle. The system receives the notification thanks to the automated workflow and runs the build automation script to evaluate the submission taking into account the test cases educator Charlie submitted and the timeliness of the submission. The system then updates the leaderboard of the battle accordingly. The team notices that they have not passed all the test cases and decide to keep working on the problem. After a few days of several teams submitting their solutions, educator Charlie checks the leaderboard to see how the students are doing.

**Goals:** G8, G9, G10. **Phenomena:** W2, W3, W4, SP6, SP13, SP14.

- **6. Educator performs manual evaluations on the submissions of a battle after it has ended:**

  After the final submission deadline of educator Charlie's battle ends, he decides to manually evaluate the submissions of the students. He sets a grade for each team based on his own evaluation criteria. After that, he consolidates the results of the battle. The system updates the leaderboards taking into account the evaluation it performed and the evaluations from educator Charlie. It then also updates the tournament leadeaboard accordingly. The system then notifies all the students participating in the battle that the final results have been consolidated. Bob, Danielle and Eve check the leaderboard and see their score. They then check the leaderboard of the tournament and see that their score has been added to the sum of all the other battles they have participated on.

  **Goals:** G11, G12, G13. **Phenomena:** W5, SP7, SP8, SP12, SP13, SP15, SP16.

- **7. Educator closes a tournament:**

  After all the battles of the tournament have ended, educator Adam decides to close the tournament. The system notifies all the students participating in the tournament that it has ended. Bob, Danielle and Eve check the leaderboard of the tournament and see their final score.

  **Goals:** G14, G15. **Phenomena:** SP9, SP17.

### 2.1.2 Domain Class Diagram

In Figure 1 we can see the Domain Class Diagram of the system.

In it we can see the following classes:

- **Student:** This class represents a student of the platform. It has an id, a name, a github account and it associates to a set of TournamentRegistrations and StudentTeams.
- **TournamentRegistration:** This class represents a registration of a student to a tournament. It saves the score the associated student has on the associated tournament. Serves as a Many to Many relationship between Student and Tournament.
- **StudentTeam:** This class represents a team of students. There can be 1 or more students in one StudentTeam. It has an id and saves the score of the team on the battle the team belongs to.
- **Tournament:** This class represents a tournament. It has an id and a registration deadline. It is associated to a set of Battles and TournamentRegistrations. It also has an owner which is an Educator and a set of invited educators.
- **Battle:** This class represents a battle. It has an id, a registration deadline, a final submission deadline, and a minimum and maximum number of participants per group. It belongs to a tournament and it contains a set of StudentTeams, a CodeKata and a GitHubRepo.
- **CodeKata:** This class represents a code kata. It has a programming language, a description, a set of test cases and a set of build automation scripts. These will be uploaded to the GitHubRepo of the battle.
- **GitHubRepo:** This class represents a GitHub repository. It has an id and it belongs to a battle.
- **GitHubIntegration:** This class represents the integration of the system with GitHub. It could act as a facade for the GitHub API.
- **Educator:** This class represents an educator of the platform. It has an id, a name, and it associates to a set of Tournaments and Battles. It can associato to tournaments as an owner or as an invited educator.

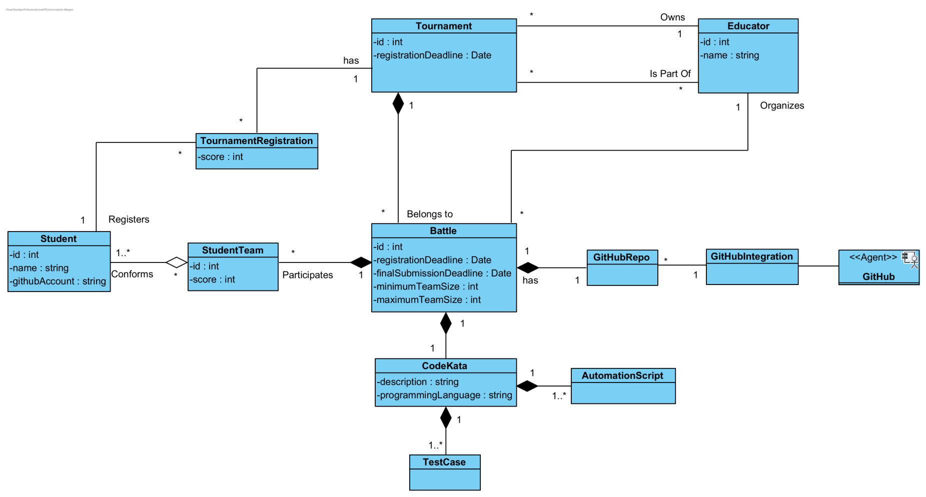

Figure 1: Domain Class Diagram

### 2.1.3 State Diagrams

Here it is possible to see the state diagrams for the classes of the system where this is more relevant.

- **Battle:**

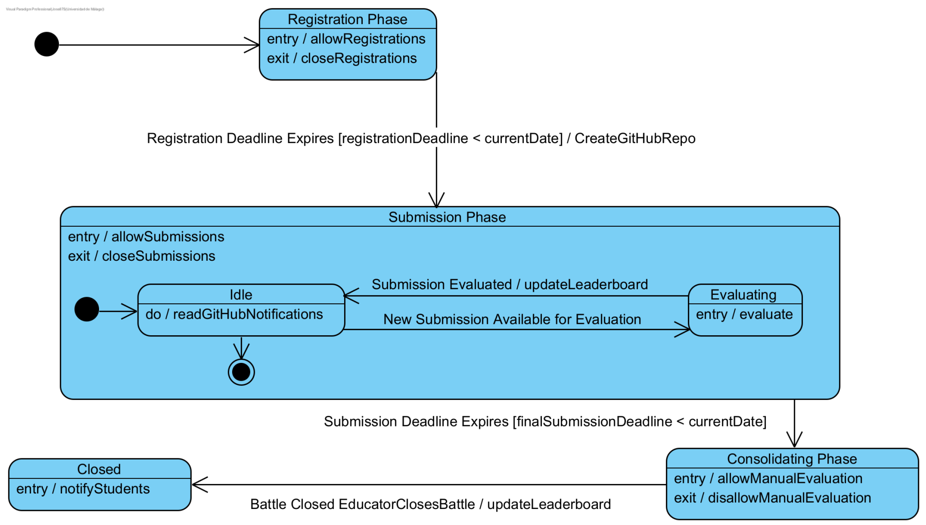
Registration Phase
entry / allowRegistrations
exit / closeRegistrations
Registration Deadline Expires [registrationDeadline < currentDate] / CreateGitHubRepo
Submission Phase
entry / allowSubmissions
exit / closeSubmissions
Idle
do / readGitHubNotifications
Submission Evaluated / updateLeaderboard
New Submission Available for Evaluation
Evaluating
entry / evaluate
Submission Deadline Expires [finalSubmissionDeadline < currentDate]
Consolidating Phase
entry / allowManualEvaluation
exit / disallowManualEvaluation
Battle Closed EducatorClosesBattle / updateLeaderboard
Closed
entry / notifyStudents

Figure 2: Battle State Diagram

A battle is firstly on it's **Registration Phase** where StudentTeams can register to it. Once the registration deadline is reached, the battle moves to the **Sub-**

**mission Phase** and it creates the **GitHubRepo**. Here the **StudentTeams** can submit their solutions to the battle through GitHub. In this state there are two substates: **Idle** and **Evaluating**. In the **Idle** state, the system is waiting for a notification from GitHub that a commit has been pushed to the main branch of one of the forked repositories of the battle. Once the system receives the notification, it moves to the **Evaluating** state where it runs the build automation scripts to evaluate the submission. Once the evaluation is done, the system updates the leaderboard of the battle and moves back to the **Idle** state. Once the final submission deadline is reached, the battle moves to the **Consolidation Phase** where the **Educator** that created the battle can perform manual evaluations on the submissions of the **Student**s. Once the **Educator** consolidates the results of the battle, the battle moves to the **Closed** state updating the leaderboard of the battle with the **Educator**'s evaluations. Here the system notifies the **Student**s that the final results have been published.

- **Tournament:**

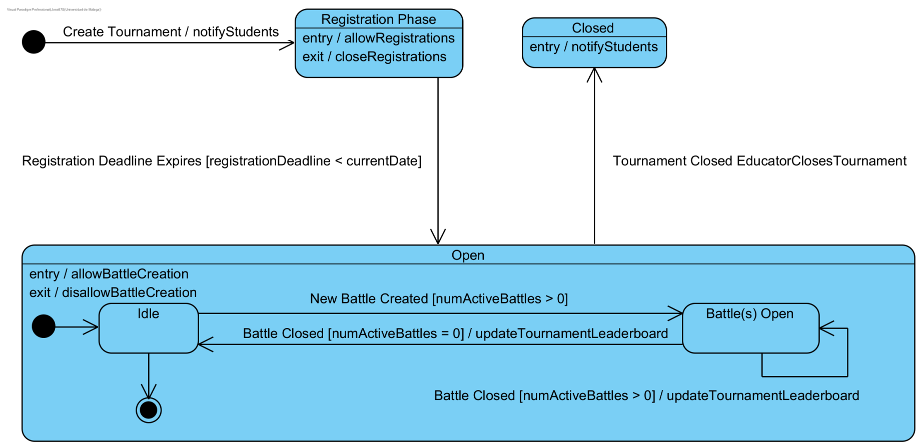
Create Tournament / notifyStudents
Registration Phase
entry / allowRegistrations
exit / closeRegistrations
Closed
entry / notifyStudents
Registration Deadline Expires [registrationDeadline < currentDate]
Tournament Closed EducatorClosesTournament
Open
entry / allowBattleCreation
exit / disallowBattleCreation
Idle
New Battle Created [numActiveBattles > 0]
Battle(s) Open
Battle Closed [numActiveBattles = 0] / updateTournamentLeaderboard
Battle Closed [numActiveBattles > 0] / updateTournamentLeaderboard

Figure 3: Tournament State Diagram

When a **Tournament** is created, all students are notified. A **Tournament**, similar a **Battle**, has a **Registration Phase** where **Student**s can register to it. Once the registration deadline is reached, the **Tournament** moves to the **Open** state. Here all the **Educator**s that are part of the **Tournament** can create **Battle**s. The **Open** state has two substates: **Idle** and **Battle(s) Open**. In the **Idle** state, there are no active **Battle**s in the **Tournament**. Once an **Educator** creates a battle, the **Tournament** moves to the **Battle(s) Open** state. Each time a battle ends, the **Tournament** leaderboard is updated. The **Tournament** moves back to the **Idle** state once there are no active **Battle**s. The **Tournament** can move to the closed state once the owner of the **Tournament** decides to close it and there are no active **Battle**s. Here, **Student**s are notified that the final results are available.

## 2.2 Product functions

- **Sign up and Login:**

  This is a basic functionality of the system. It allows for students and educators to sign up and login to the system. When signing up, the user must provide a name, an email, a password and a GitHub account. The system will then send a confirmation email to the user. Once the user confirms their email, they can login to the system.

- **Create Tournament:**

  This functionality allows for educators to create tournaments. When creating a tournament, the educator must provide a registration deadline and can invite other educators to the tournament. Once the tournament is created, the system will notify all the students that a tournament has been created.

- **Register to Tournament:**

  This functionality allows for students to register to tournaments. When registering to a tournament which is on its registration phase. Students are able to join all the tournaments which are on this phase.

- **Create Battle:**

  This functionality allows for educators to create battles. When creating a battle, the educator must provide a registration deadline, a final submission deadline, a minimum and maximum number of participants per group, a programming language, a description of the problem, a set of test cases and a set of build automation scripts. Once the battle is created, the system will notify all the students participating in the tournament that a battle has been created. The system will use the files provided by the educator to create a GitHub repository for the battle.

- **Register to Battle:**

  This functionality allows for students to register to battles. When registering to a battle, which is on its registration phase, students can invite other students or accept invitations from other students to join the battle, always taking into account the minimum and maximum number of participants per group set by the educator.

- **Evaluate submissions:**

  This functionality allows for the system to evaluate the submissions of the students. Each time the system receives a notification from GitHub that a commit has been pushed to the main branch of one of the forked repositories of the battle, the system will pull the latest sources, run the tests and update the leaderboard of the battle accordingly.

- **Consolidate results:**

Once the final submission deadline of a battle ends, the educator that created the battle can consolidate the results of the battle. This means that the educator can manually evaluate the submissions of the students and set a grade for each team, which will be combined with the evaluation performed by the system. Once the educator finishes consolidating the results, the system will update the leaderboard of the battle and notify all the students.

- **Close Tournament:**

  Once all the battles of a tournament have ended, the owner of the tournament can close it. This means that the system will update the leaderboard of the tournament and notify all the students that the tournament has ended.

## 2.3 User characteristics

- **Students:** They participate in battles and tournaments to improve their software development skills. They can form teams, work on the battles, and submit their solutions through GitHub.
- **Educators:**: They are responsible for creating and managing battles and tournaments. They set various parameters for battles, evaluate the solutions submitted by students, and provide feedback.

## 2.4 Assumptions, dependencies and constraints

### 2.4.1 Domain Assumptions

- **D1:** All users have a valid email address and can access their email to confirm their registration.
- **D2:** GitHub integration is reliable, and the system can communicate with GitHub to create repositories, pull code, and receive notifications.
- **D3:** Students and educators have a basic understanding of version control systems, especially GitHub.
- **D4:** Educators have the necessary expertise to create meaningful and effective code kata battles, including defining appropriate test cases.
- **D5:** Students have the required technical skills to participate in code kata battles, including forking repositories, setting up GitHub Actions, and writing code.
- **D6:** The system has access to resources (computational resources, storage, etc.) to perform automated evaluations of code submissions.
- **D7:** Users have a GitHub account.
- **D8:** The educator's uploaded description of the problem, programming language test cases and build automation scripts are all related to each other.
- **D9:** Educators have a set of evaluation criteria through which they perform their manual evaluations.

### 2.4.2 Dependencies

- The system relies on GitHub for version control, code repository management, and continuous integration through GitHub Actions.
- Email services are required for user registration confirmation and communication.

### 2.4.3 Constraints

- The system's functionality heavily depends on the reliability and availability of external services such as GitHub and email services.
- The system's performance is constrained by the response time of GitHub services for repository creation and code evaluation.
- Users must have internet access to interact with the CKB platform and GitHub services.

# 3 SPECIFIC REQUIREMENTS

## 3.1 External Interface Requirements

### 3.1.1 User Interfaces

### 3.1.2 Hardware Interfaces

### 3.1.3 Software Interfaces

### 3.1.4 Communication Interfaces

## 3.2 Functional Requirements

## 3.3 Performance Requirements

## 3.4 Design Constraints

### 3.4.1 Standards compliance

### 3.4.2 Hardware limitations

### 3.4.3 Any other constraint

## 3.5 Software System Attributes

### 3.5.1 Reliability

### 3.5.2 Availability

### 3.5.3 Security

### 3.5.4 Maintainability

### 3.5.5 Portability

# 4 FORMAL ANALYSIS USING ALLOY

# 5 EFFORT SPENT

# 6 REFERENCES