ETSA03-ADM-INS

Project Instructions for the LU Rumble

Version 1.0

Prepared by Markus Borg
Dept. of Computer Science, Lund University

April 3, 2020

Contents

Revision History

Name	Date	Change description	Version
Markus Borg	2017-12-07	Initial draft.	0.1
Markus Borg	2017-12-27	Complete draft of intro, engineering, monetizing, and strategizing. Sent for external review.	0.2
Markus Borg	2017-12-31	First draft of sprints.	0.3
Markus Borg	2018-01-02	First complete draft.	0.4
Markus Borg	2018-02-05	Updated after internal review.	0.5
Markus Borg	2018-03-06	Updated after external review.	0.6
Markus Borg	2018-03-08	Minor consistency improvements.	0.7
Markus Borg	2018-03-24	Updated after course start. Added robot communication, releases and deliverables.	0.8
Markus Borg	2018-03-29	Replaced test specification with STS. Fixed L4 dates. Introduced RoboTalk.	0.85
Markus Borg	2018-04-18	Clarified the discount rule for Robot Market.	0.86
Markus Borg	2018-04-21	Added the Lab4 tools, and details for Sprint 2 and the Beta Release.	0.87
Markus Borg	2018-04-21	Added details for Sprint 3 and the Final Release.	0.88
Markus Borg	2018-04-24	Added review protocol in the process figure.	0.89
Markus Borg	2018-05-05	Updated ETSA02 RoboTalk to v1.3.	0.90
Markus Borg	2018-05-05	Clarified Monetization in two parts, updated Strategizing.	0.91
Markus Borg	2018-05-05	Updated LU Rumble rules and awards.	0.92
Markus Borg	2018-05-07	Clarified bundle deals.	0.93
Markus Borg	2018-05-10	Corrected L8 date, fixed typos.	0.94
Markus Borg	2019-02-05	Basic updates for 2019 edition.	0.95
Markus Borg	2019-03-04	Detailed updates for 2019 edition.	0.96
Markus Borg	2019-03-25	Final touch for 2019 edition.	0.97
Markus Borg	2019-03-30	Added instructions for time reporting.	0.98
Markus Borg	2019-04-13	Clarified expectations on deliverables.	0.981
Markus Borg	2019-04-16	Postponed Final release.	0.982
Markus Borg	2019-05-06	Replaced web page with display window.	0.983
Markus Borg	2019-05-09	Clarified bundle deals for Droids.	0.984
Markus Borg	2020-03-21	Updates for the 2020 online edition.	0.985
Markus Borg	2020-04-03	Updates reflecting new course code and CI.	1.0

1 Introduction

This document describes the project in ETSA03 Introduction to Software Engineering – Methodology. Together with other students, you will work together to engineer a robot for competition in LU Rumble – a local instance of team battles in Robocode, a programming game in which robots compete in battles without human intervention by executing their programs.

In the remainder of this document, we use the term "group" to refer to the students working together during the project. While the student constellations would better be described as teams, i.e., groups of people working together toward a common goal, we reserve the term "team" to robots competing together in Robocode.

This document is organized as follows: Section 1.1 presents the learning goals of the project in relation to the ETSA03 formal course description.

1.1 Learning goals

The project primarily aims to increase your ability to develop high quality software systems using established software engineering best practices. Moreover, basic software business concepts will be introduced as each project group will offer their product on a highly competitive market.

The project complements the theoretical concepts introduced during the lectures by taking a practical approach to the presentation of fundamental software engineering concepts such as specification, version control, continuous integration, testing, sprints, and releases. By the end of the course, you will have acquired new skills with essential components of the contemporary software engineering tool chain: the Java programming language, the Eclipse integrated development environment, the JUnit testing framework, the git distributed version control system, GitHub cloud-based project hosting, the Gradle build system, Travis CI with modern code analysis tools such as CodeClimate and CodeScene.

On top of your skills with traditional software engineering concepts and the tool chain, you will address aspects of market-driven software engineering. Each project group will practice making critical business and engineering decisions in a controlled fashion. The course introduces several important activities in software business, including analyzing the business ecosystem, finding a niche market, marketing a product, competitive pricing strategies, and customer negotiations.

More formally, the project is primarily designed to meet the practical requirements of the ETSA03 course description, i.e., the "Competencies and Skills". However, the project will also reinforce the theoretical concepts listed under "Knowledge and Understanding" and it contributes to the "Judgment and Approach" section.

- Competencies and Skills
  - Develop a basic business plan for a software product
  - Market a software product
  - Identify, analyse and specify requirement for a software product
  - Manage a development environment for programmers
  - Contribute to a development team's source code
  - Develop basic automated tests
  - Write technical documentation
  - Perform reviews of technical documentation and source code
  - Collaborate in a development team
- Knowledge and Understanding
  - Be able to define basic terms and definitions in software engineering.
  - Be able to describe the most common software development processes.
  - Be able to explain the most important steps in requirements engineering
  - Be able to explain how software testing is conducted.
  - Be able to describe what a software architecture design is.
  - Be able to describe the most important steps in project planning and project tracking.
  - Be able to discuss software products from a life cycle perspective
  - Be able to explain the most important parts of a business plan for software products
  - Be able to explain why correct technical documentation is important
- Judgment and Approach
  - Understand the complexity involved in developing software systems
  - Have an understanding of the professional role of the engineer

1.2 Robocode – Build the best, destroy the rest!

The goal of Robocode is to implement the behavior of a robot to compete against other robots in a battle arena. The contestants have no direct influence on the game, instead they write the AI of the robot telling it how to behave and react on events occurring in the battle arena. Battles are running in real-time and on-screen, see Figure 1.1. Robocode battles, referred to as "rumbles", are either in the form of duels between individual robots, free-for-all battles with multiple robots, or battles between robot teams.

From `http://robocode.sourceforge.net/docs/ReadMe.html`: Robocode offers a complete development environment, and comes with its own installer, built-in robot

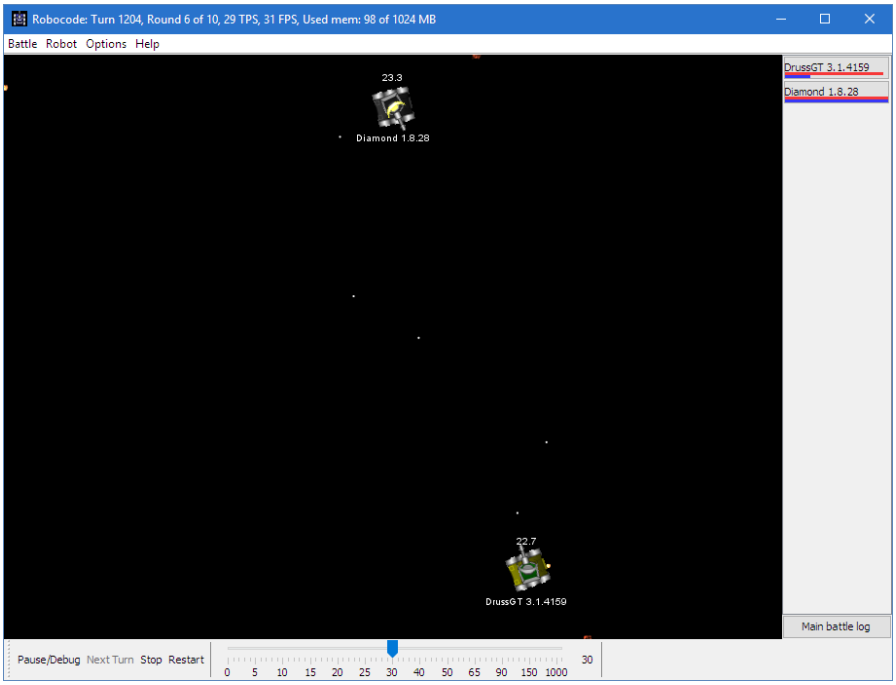

Figure 1.1: Screenshot from the Robocode battle arena. (Image credit: robowiki.net user MultiplyByZer0)

editor and Java compiler. Robocode only pre-requires that a JVM (Java Virtual Machine) to exist already on the system where Robocode is going to be installed. Hence, everything a robot developer needs to get started is provided with the main Robocode distribution file (`robocode-xxx-setup.jar`). Robocode also supports developing robots using external IDEs like Eclipse.

Robocode is an Open Source project, which means that all sources are open to everybody. In addition, Robocode is provided under the terms of EPL (Eclipse Public License). The Robocode game was originally started by Mathew A. Nelson as a personal endeavor in late 2000 and became a professional one when he brought it to IBM in July 2001. In the beginning of 2005, Mathew convinced IBM to release Robocode as Open Source on SourceForge. Eventually, Flemming N. Larsen took over the Robocode project at SourceForge as administrator and developer in July 2006 to continue the original Robocode game – which now is hosted on GitHub.

1.3 Project overview

Each project group will develop a robot using established software engineering practices. Furthermore, each group will compose a robot team to compete in a "LU Rumble" at the final lecture in the course. However, no project group is allowed to field their own robot – instead robots developed by other groups must be purchased on an open market

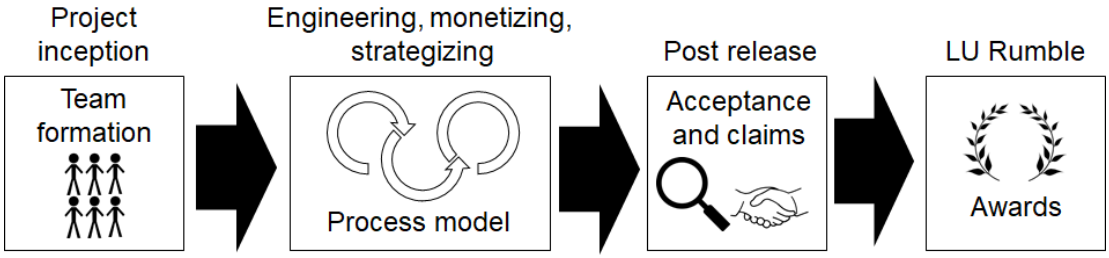

Figure 1.2: The four phases of the project. Expect also a post mortem phase, concluding with the take-home exam.

(in truth, a somewhat regulated market). Consequently, each project group has two primary goals: 1) maximizing profit by selling a successfully engineered robot on the market and 2) winning the LU Rumble by composing a competitive Robocode team. Project groups pursue the two goals by completing three main activities that we refer to as *strategizing*, *engineering*, and *monetizing*, respectively.

Figure 3.1 shows the four main phases of the project. First, during the course inception, the course infrastructure and the project tasks are introduced. More importantly, project groups consisting of six students (preferably!) are established. Second, the backbone of the course follows: three development sprints mixing engineering, monetizing, and strategizing. Third, purchasers of robots perform acceptance testing to ensure that the delivered robot fulfills the expectations – otherwise purchasers file business claims to request money back. Fourth, in the last phase of the project, the LU Rumble takes place followed by an awards ceremony to recognize the winning robot team – and the most profitable project groups.

Figure 1.3 shows the context of each individual group during the project. Organizationally, your group has three divisions: 1) Engineering, 2) Executive, and 3) Sales. Everyone in the group will shoulder several roles, as you all will be involved in the Engineering division as developers and testers – supported by the mandatory lab sessions. Figure 1.3 indicates this by the six small figures denoted "Dev/Test". On top of this, your group will also assign the following additional roles: 1) project manager (PM), 2) requirements engineer (RE), 3) software architect (SA), 4) quality engineer (QE), 5) configuration manager (CM), and 6) sales engineer (SE). Each role is described in detail in Section 4.1. If there are only five people in a group, the role as configuration manager is left out.

Project groups do not exist in isolation, instead they act in a business ecosystem. During engineering of the robot, each project group will be part of two supplier-customer relationships. First, your sales engineer will be responsible for promoting sales of the robot under development. The project manager of another project group will sign a contract of sale, establishing a formal relationship. Your sales engineer will be the primary communication point for the purchasing group's project manager, i.e., a supplier-customer relationship involving feedback, feature requests, and negotiations. Second, analogously, your project manager will act as the customer in a supplier-customer relationship with

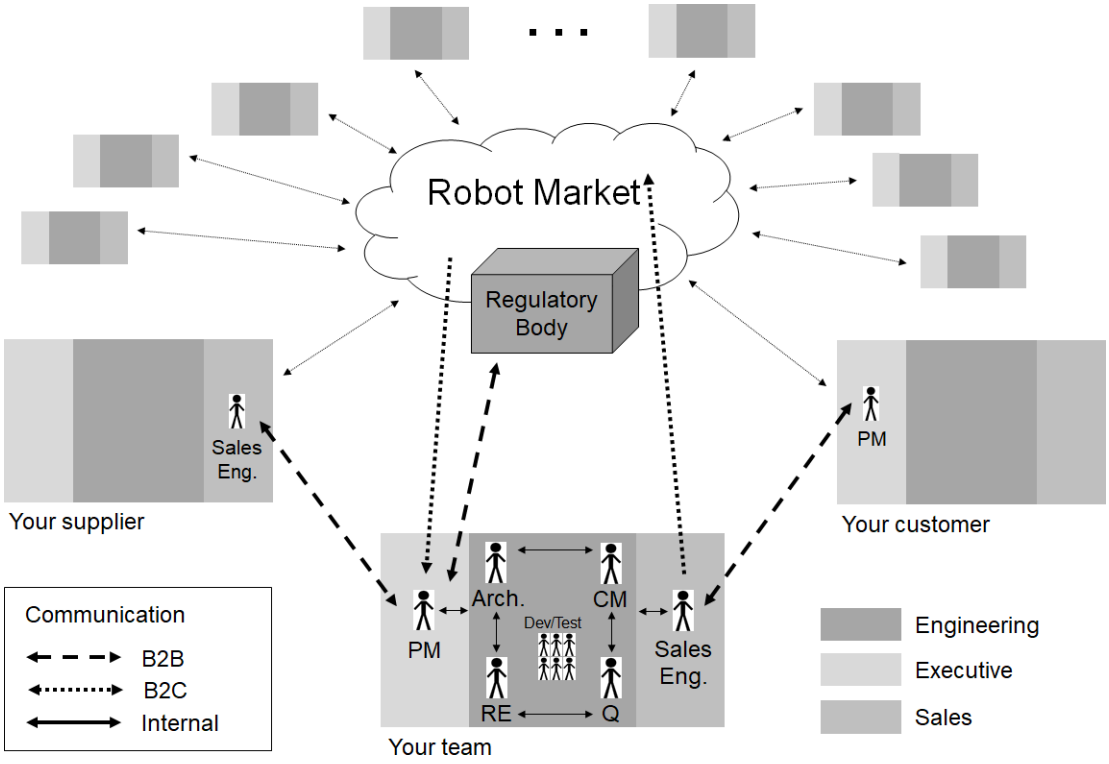

Figure 1.3: The project context. Your project group will interact with your supplier's sales engineer, your customer's project manager, the open Robot Market, and the regulatory body.

another project group. Note that, in some cases, project groups might buy from and sell to the same other group – this is not regulated in the ecosystem.

On top of the two supplier-customer relationships, your project group will also communicate with the regulatory body (represented by your project supervisor). Your project group will hand in various signed contracts to the regulatory body, turn to it to get support during supplier-customer negotiations, and to report the results from your acceptance testing. The regulatory body might also contact you with regulatory changes during the course of the project. Finally, you will practice one-way communication with the Robot Market, i.e., the open market where you will sell your robot. All project groups will complement their robot teams with additional robots during the last week of the project – and your sales engineer will be responsible for maximizing sales of your robot during this period.

2 The project from a bird's eye view

The project consists of three main activities, interconnected as presented in Figure 2.1. This section presents an overview of the three activities.

2.1 Engineering the robot

Each project group will develop a robot (see Figure 2.2) according to established engineering principles. These principles include product prototyping, integrated requirements engineering, and automated testing. The process model is organized in three development sprints, further described in Section 3.2. The engineering process is dominated by three core activities: *specification*, *construction*, and *verification*. In line with Agile development methodologies, all sprints consist of a mix of these three activities. Still, the groups' focus during the project will progress from specification, through construction, to verification – also reflected by the order of the lectures.

2.1.1 Specification

Requirements engineering, i.e., a systematic approach to developing a product with an end-user perspective, will be an integrated activity throughout the project. The requirements will be captured in a Software Requirements Specification (SRS) adhering to a template structure provided by the Institute of Electrical and Electronics Engineers (IEEE). IEEE is the world's largest technical professional organization, and a natural resource for software engineers – possibly contented by the Association of Computing Machinery (ACM). Both organizations support engineers in development of software products adhering to state-of-practice approaches and provide ethical guidelines.

The IEEE SRS template provides a standard document structure: 1) an introduction section, 2) an overall description of the product under development, and 3) a list of product features further broken down into detailed requirements. Furthermore, the template supports engineers to specify quality requirements such as performance, maintainability, and memory footprint. Project groups are recommended to use a combination of traditional requirements of "shall" format and step-wise use case descriptions. While also user stories are covered in the lectures, it uses a narrative style that is less appropriate for development of autonomous robots.

The SRS connects all roles in the project group. While the requirements engineer is responsible for the SRS, the sales engineer is responsible for developing a sales pitch based on its content. The SRS will be public on the Robot Market, i.e., when other project groups consider purchasing your robot, they will critically review your SRS. The project manager will double as a Robocode domain expert, thus having the best

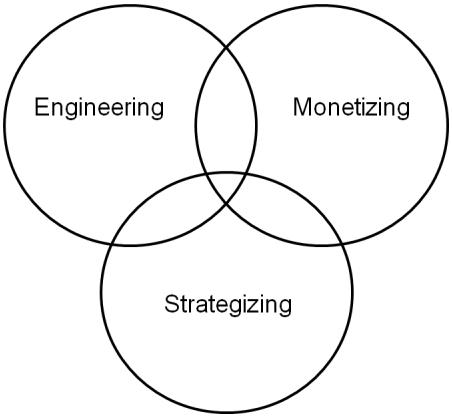

Figure 2.1: Project components.

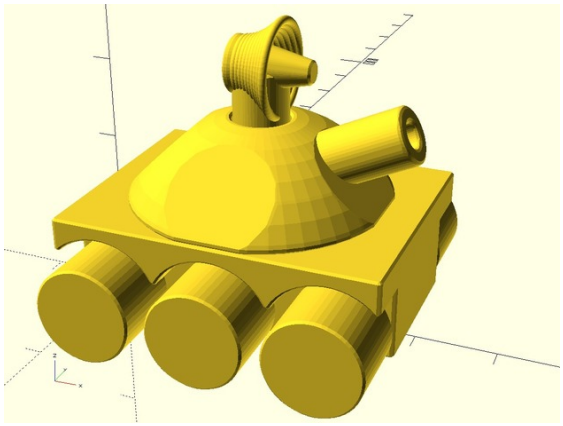

Figure 2.2: Model of the Robocode robot (© Klaus Knopper under CC BY-SA 3.0).

chance to predict what features will be valued on the Robot Market, and will provide important input to the SRS – effectively acting as a proxy-customer until the supplier-customer relationship has been established. Finally, the software architect is responsible for implementing the SRS in source code and the quality engineer is responsible for verifying that the developed product fulfills the SRS.

There are several quality attributes that a high-quality SRS should comply with. First, the SRS should be complete, i.e., if properly implemented, the developed product shall be a competitive robot. Just as importantly, the robot shall not offer any features that are not properly captured in the SRS – the customer is not allowed any surprises. Second, the SRS should constitute a coherent document that is easy to follow. Furthermore, each individual requirement should be:

Complete All information needed to implement source code to fulfill the requirement shall be specified. Developers must not be forced to read between the lines.

Consistent There are no contradictions between individual requirements.

Feasible The requirement can be practically implemented by the developers given the available resources. Also, the requirement will bring value to the end user or another stakeholder.

Modifiable An SRS is rarely static, thus all requirements must be able to change. Each requirement should be clearly identifiable.

Unambiguous A requirement shall only be interpretable in one way, i.e., no subjectivity can exist. Quality requirements should be quantified, not described using vague adjectives.

Design free Requirements shall specify what the system must or must not do, but not how the software will ensure the requirement is met – leave that to the construction phase.

Testable Requirements shall be possible to verifiable, preferably by testing, otherwise inspection.

As a result from the specification activities, each project group shall produce the following deliverables, further described in Section 3.2:

Sprint 1 SRS v0.5 – Main features captured

Sprint 2 SRS v0.9 – Detailed requirements specified

Sprint 3 SRS v1.0 – A complete Robot specification

2.1.2 Construction

The first two computer labs will help project groups set up Robocode development in Eclipse and source code control in git. Moreover, the second computer lab will initiate automated unit testing, an activity that overlaps construction and verification.

An early construction activity is software design. In this project, the design is limited to object-oriented design – the robot development is too restricted to address any considerations on the software architecture level and there is also no user interaction design. Each project group will present the Object-Oriented Design (OOD) using a Unified Modeling Language (UML) class diagram. As the project is inspired by Agile development methodologies, there will be no big design activity upfront. Instead, the OOD will evolve as the construction phase proceeds. The software architect is responsible to complement the final source code delivery with a class diagram to describe the OOD after-the-fact. Any project group that recognizes a value in using class diagrams in their robot sales pitch are of course welcome to do so. Furthermore, the final source code deliverable shall be complemented by Javadoc documentation (excluding the test code).

As most practical construction work will be done outside of scheduled sessions, how the backbone of the programming will be conducted must be decided by the individual project groups. Still, for this project, we recommend groups to follow a handful of well-known Agile development practices, originally proposed as part of Extreme Programming (XP). Below we list five that we believe are particularly suitable for the process model, along with their Wikipedia definitions:

Pair programming All code is produced by two people programming on one workstation. One programmer has control over the workstation and is thinking mostly about the coding in detail. The other programmer is more focused on the big picture, and is continually reviewing the code that is being produced by the first programmer. Programmers trade roles after minute to hour periods. The pairs are not fixed; programmers switch partners frequently, so that everyone knows what everyone is doing, and everybody remains familiar with the whole system, even the parts outside their skill set. This way, pair programming also can enhance team-wide communication and goes hand-in-hand with the concept of Collective Code Ownership.

Continuous integration The development team should always be working on the latest version of the software. Since different team members may have versions saved locally with various changes and improvements, they should try to upload their current version to the code repository every few hours, or when a significant break presents itself. Continuous integration will avoid delays later on in the project cycle, caused by integration problems, aka. "big bang integration". In the projects, you will use a continuous integration service provided by Travis CI.

Code refactoring Restructuring existing code without changing its external behavior. Refactoring improves quality attributes of the software, e.g., code readability,

performance, maintainability. Refactoring is often initiated when a so called "code smell" is identified, i.e., structures in the code that indicate violation of fundamental design principles and negatively impact design quality.

Test-driven development Create unit tests before the eventual code is written. This approach is intended to stimulate the programmer to think about conditions in which his or her code could fail. Test driven development proceeds by quickly cycling through the following steps: 1) write unit tests, 2) run tests, watch them fail, 3) write code, 4) run tests, watch them pass, and 5) refactor the code. Adhering to test-driven development leads to a large set of test cases that can be automatically executed, increasing developers' confidence when e.g. refactoring code.

Collective code ownership Everyone is responsible for all the code; therefore, everybody is allowed to change any part of the code. Collective code ownership is not only an organizational policy but also an attitude. Pair programming, especially overlapping pair rotation, contributes to this practice: by working in different pairs, programmers better understand the system context and contribute to more areas of the code base. Collective code ownership may accelerate development because a developer who spots an error can fix it immediately, which can reduce bugs overall. However, programmers may also introduce bugs when changing code that they do not understand well. Sufficiently well-defined unit tests should mitigate this problem: if unforeseen dependencies create errors, then when unit tests are run, they will show failures.

The construction phase involves prototyping to quickly create a working robot. Prototyping supports specification activities by enabling early executable software – truly beneficial in contexts where the requirements are very unstable. Already at the end of sprint 1, around the LU Robot Fair, the project group shall deliver a first prototype of the robot. The prototype will help the purchaser to understand your product vision, and enable better feedback. The second deliverable shall be a Minimum Viable Product (MVP) inspired by lean software development practices, i.e., a product with just enough features to satisfy early customers and to stimulate feedback for future robot evolution throughout Sprint 3.

As a result from the construction activities, each project group shall produce the following deliverables. Each release shall be prepared as two jar-files: 1) one jar-file for customers and 2) one jar-file with embedded source code for the regulatory body. As previously stated, the final release to the regulatory body shall be complemented by a class diagram and Javadoc documentation, as further described in Section 3.2:

Sprint 1 Robot Alpha release (Robot v0.5) – A prototype

Sprint 2 Robot Beta release (Robot v0.9) – An MVP

Sprint 3 Robot Final release (Robot v1.0) – The final robot

2.1.3 Verification

All computer labs but the first one will involve working with the JUnit framework for automated testing. Project groups will work with unit testing as a natural companion to the programming during development. While JUnit is designed for automated execution of unit test cases, groups will also use JUnit for system testing of both functional and non-functional (quality) aspects of the robot under development. Finally, groups will learn to use measure code coverage and use static code analysis tools to support software quality assurance.

The quality engineer is responsible for delivering three growing development artifacts during the projects, as well as a final test report at the final release. First, a suite of automated unit test cases shall evolve during the project. At sprint 1, an initial version shall simply be executable, but at the final release all source code shall be exercised by unit tests (note that the software architect is responsible for the implementation of the unit test cases). Second, a suite of automated system test cases shall be developed – an initial version at sprint 2, and the complete suite with the final release. Third, a Software Test Specification (STS) that describes the project group's test and quality activities. At sprint 3, all requirements in the SRS shall be verified either by automated test cases, manual test cases, or targeted by a convincing argumentation that the requirement has been verified through inspection.

Verification activities are naturally supported by tools. Apart from the JUnit framework, the project groups will use the Robocode framework RobotTestBed to automate system testing, and EclEmma to measure code coverage. Furthermore, the project groups will learn about source code quality assurance by running two static code analysis tools: SpotBugs and SonarLint. The tools automatically analyze the source code from complementary quality perspectives.

The final robot release shall be accompanied with a test report, signed by the quality engineer, showing that all test cases have been executed – and all corresponding test verdicts. The customer will surely expect that all test cases passed, i.e., that you provide evidence that the entire SRS has been fulfilled by the final robot. In addition to the test verdicts, the test report shall contain appendices with extracted report from the code coverage tool as well as the static code analysis tools.

As a result from the verification activities, each project group shall produce the following deliverables, further described in Section 3.2:

Sprint 1 STS v0.5 – Initial document structure

Sprint 2 STS v0.9 – Unit testing and system testing partly specified

Sprint 3 STS v1.0 – A complete specification (complemented by a signed test report)

2.1.4 Robot-to-Robot communication

Robots that extend TeamRobot can communicate with team members, as specified in the Robocode API: `http://robocode.sourceforge.net/docs/robocode/robocode/TeamRobot.html`

Serializable objects can be sent between robots, e.g., point objects containing coordinates, color objects to set uniform team colors, and message strings with orders from the leader. The following three methods represent the backbone of robot communication.

Broadcasting message to all team members:

```
public void broadcastMessage(Serializable message)
```

Sending messages to one specific team member:

```
TeamRobot.sendMessage(String name, Serializable message)
```

When a robot receives a message, the following method is called:

```
onMessageReceived(MessageEvent)
```

The MessageEvent always contains the name of the sending robot and a specific message. As stated above, the message can be instances of various classes. Please refer to the Robocode Sample Team to see an example implementation, or refer to the ETSA03 GitHub repository.

The only type of communication that must be implemented by the robots entering LU Rumble is team color – this is mandated by the Regulatory Body. During the LU Rumble, all robots of the same team must share the same color scheme. With 10 robots battling at the same time, it is critical to help spectators understand the action on the battlefield. This introduces one of the following regulatory requirements to all robots:

- For normal robots and droids:
  "The robot shall set its color scheme upon request by a leader bot according to ETSA03 RoboTalk v1.X."
- For leader bots:
  "The robot shall set its color scheme at the start of the battle, and request the other team robots to follow suit according to ETSA03 RoboTalk v1.X."

ETSA03 RoboTalk also specifies a string-based communication protocol. Teams are free to implement as much of the communication protocol as they find useful. While it is completely optional, it might be expected by potential customers on Robot Market. Please refer to the classes `MessageReader` and `MessageWriter` on the ETSA03 GitHub repository for an implementation suggestion. A message string should have the following overall structure, including the new lines.

```
[0-1] leadership;[followMe|leadMe]
[0-1] teamMode;[offensive|defensive]
[0-1] myPos;x;y
[0-*] friendPos;name;x;y
[0-*] enemyPos;name;x;y
[0-*] enemyDetails;name;x;y;velocity;energy;heading;gunHeading
```

```
[0-1] bullet;x;y;absBearing;bulletPower
[0-1] targetEnemy;name
[0-1] targetPos;x;y
[0-1] moveTo;x;y
```

Example message:

```
teamMode;offensive
myPos;100;100
enemyPos;bot1;50;50
enemyPos;bot2;200;150
targetPos;50;50
```

The commands shall have the following meaning:

leadership Tells the receiving robot(s) that the sender takes on leadership (followMe) or reports as a follower (leadMe).

teamMode Tells the receiving robot(s) that the team shall enter offensive mode (offensive) or defensive mode (defensive).

myPos Tells the receiving robot(s) its current position on the 2D battlefield.

friendPos Tells the receiving robot(s) the position of an identified robot in the same team.

enemyPos Tells the receiving robot(s) the position of an identified enemy robot.

enemtDetails Tells the receiving robot(s) detailed information about an enemy robot.

bullet Tells the receiving robot(s) that a bullet has been fired and details its trajectory.

targetPos Tells the receiving robot(s) to fire at a specific position on the 2D battlefield.

targetEnemy Tells the receiving robot(s) to target a specific enemy robot.

moveTo Tells the receiving robot(s) to move to a specific position on the 2D battlefield.

ETSA03 RoboTalk might evolve during the course, i.e., the Regulatory Body will possibly release new versions. All groups are welcome to provide feedback and requests to the standardization work done by the Regulatory Body. To do so, the project manager should submit a "Change Request" to the Regulatory Body. The Change Request should carefully describe what changes are proposed, and motivate why this request should be considered.

ETSA03 RoboTalk is version controlled, and student groups that decide to implement communication should carefully specify in their SRS which version of the protocol is implemented. For further details and a formal specification, including both communication of color and string commands, refer to the latest version of ETSA03 RoboTalk on Google Drive.

	Robot A	Robot B	Robot C	Robot D	Robot E	Robot F	Robot G
Group A	N/A	€ 10	€ 10	**€ 60**	€ 20	€ 10	€ 10
Group B	€ 10	N/A	**€ 30**	€ 22	€ 35	€ 30	€ 15
Group C	**€ 10**	€ 10	N/A	€ 30	€ 22	€ 25	€ 10
Group D	€ 10	**€ 15**	€ 29	N/A	€ 32	€ 31	€ 10
Group E	€ 10	€ 10	€ 30	€ 23	N/A	€ 12	**€ 30**
Group F	€ 10	€ 10	€ 28	€ 17	**€ 40**	N/A	€ 30
Group G	€ 10	€ 10	€ 10	€ 27	€ 10	**€ 50**	N/A

Figure 2.3: Example of seven purchase arrays submitted to the regulatory body.

2.2 Monetizing the robot

One of the primary goals of the project is to maximize the profit of the group, i.e., monetizing the robot. The business aspects of the project involves a combination of bespoke software engineering and software engineering for an open market. All project groups will offer their robot on a highly competitive market – consisting of the other groups, each with a €100 budget to invest in a robot team for the LU Rumble. The monetization is split in two parts: 1) bespoke supplier-customer relationships and 2) selling robots on Robot Market.

2.2.1 Bespoke supplier-customer relationships

The project groups will market their Robot already at a robot fair already after Sprint 1. At this early point in time, only an Alpha release of the robot will be available and an initial version of the SRS – still, the sales engineer is responsible for developing a convincing sales pitch. All groups will get the chance to pitch their robot through a two minute video at the robot fair, and also present it online on the Robot Market, complemented by the most recent SRS.

After the robot fair, all project groups will submit a "purchase array" to the regulatory body. The purchase array consists of one element per group, and each element should contain an integer representing how much your group is willing to pay for the corresponding robot. You are not allowed to make an offer for your own robot, and you are not allowed to offer less than €10 for any other robot. The purchase array should be signed by the project manager and delivered as a physical copy to the regulatory body through the mailbox in "grå skåpet".

Figure 2.3 shows an example of purchase arrays submitted to the regulatory body. The purchase arrays are never disclosed, the regulatory body matches project groups into supplier-customer relationships. In the example, cells with bold text show winning bids. Group D, highlighted with a white row, was particularly interested in Robots E, F, and C – but unfortunately, other project groups offered more. Group D instead got to buy Robot B for €15. Note that the highest bid for a robot can be a tie, such as for Robot A and G. In the case of Robot A, no project group offered more than the minimum €10, which resulted in a (possibly unexpected) bargain buy.

When all project groups have provided their purchase arrays, the regulatory body

will commence its work to establish supplier-customer relationships. For each robot, the highest bidder turns into a customer of the group engineering the corresponding robot. If there is a tie, the customer relationship will be assigned randomly. In the end, all project groups will 1) pre-order a robot from a supplier, and 2) have a customer that offered at least €10 to pre-order a robot (the price of the cheapest basic bot). Note that the decision of the regulatory body cannot be appealed.

Once robots have been pre-ordered, 25% of their costs will be transferred from the customer to the supplier. This amount will not be refunded in any case – it represents the risks of purchasing a robot from an external organization. The remainder of the cost will be transferred at the delivery of the final robot. After the final delivery, the project manager of the purchasing project group conducts acceptance testing of the robot. If the acceptance testing shows that the delivered robot does not fully comply with the SRS, the project manager of the purchasing group is entitled to file a business claim to the regulatory body, further described in Section **??**.

2.2.2 Selling robots on Robot Market

In the Post release phase (cf. Fig. 3.1, each project group will also offer their robot on the Robot Market. Simultaneously, all project groups will seek to complement the pre-ordered (by now hopefully delivered) robot with additional robots to build a competitive robot team for the LU Rumble. Throughout Sprints 2 and 3, the sales engineer is responsible for promoting sales of the robot on Robot Market. Each group will also set a price for its robot – a group must at least ask for as much as what the bespoke customer has paid. Furthermore, project groups can (but are not required to) offer robot bundles with discounts for buyers that purchase more than one robot, i.e., bundles of two, three, or four robots. The total cost of each larger bundle most exceed the smaller bundles.

The regulatory body highlights that the bundle options are not possible for suppliers of leader bots, as each team can field only one leader robot. Also note that suppliers of droids cannot offer bundle options beyond four robots, even though droids do not count toward the team limit of five robots. Buying more than four droids is allowed, but five droids will cost the bundle deal for four robots plus the price of a single droid (or as many additional droids that are requested).

Figure 2.4 shows an example of robot prices on Robot Market. Groups A and C develop leader bots and are not allowed to offer any bundle deals. Group E, highlighted in the white row, offers a single robot for €40 supplier of Robot E and the following bundle deals: 2 robots for €60, 3 robots for €65, and four robots for €70. Group F creates a bundle deal for two robots, but decides not to offer special prices for three or four robots. No bundle deals are available for the three ETSA03 basic bots. All groups must set their prices on Robot Market no later than at L7 – no subsequent price updates are allowed.

Figure 2.5 shows an example of how two project groups purchase robots. After the robot fair, Group A strikes a deal with Group C for €55. As Robot C was quite costly, Group A complements it with two cheaper Robot B droids in a €38 bundle deal to create a team of three robots for LU Rumble. Robot B resulted from bespoke development for

	Type	1x robot	2x robots	3x robots	4x robots
Robot A	Leader bot	€ 40	N/A	N/A	N/A
Robot B	Droid	€ 20	€ 38	€ 55	€ 75
Robot C	Leader bot	€ 30	N/A	N/A	N/A
Robot D	Normal bot	€ 60	€ 80	-	-
Robot E	Normal bot	€ 40	€ 60	€ 65	€ 70
Robot F	Normal bot	€ 50	€ 75	-	-
Robot G	Normal bot	€ 30	-	-	-
Basic Leader Bot	Leader bot	€ 20	-	-	-
Basic Melee Bot	Normal bot	€ 15	-	-	-
Basic Droid	Droid	€ 10	-	-	-

Figure 2.4: Example of robot prices on the Robot Market, including bundle deals.

Group C – and they liked the result enough to buy two additional robots in a bundle deal at the later Robot Market stage. Group C must pay the price of the 3x Robot B bundle deal minus the price of a single robot (€56 - €20 – as they already bought one!). Furthermore, Group C complements the cheap droids by two basic bots: a Basic Leader Bot and a Basic Bot. Both Groups A and C spend €93 of their budgets. The remaining €7 and €9 are counted towards the teams' profits, increasing their chances to win the Most Profitable Team award (see Section 3.4).

2.3 Strategizing to win the LU Rumble

The LU Rumble is a customized instance of a Robocode team rumble. Each project group will compose a team of robots that will compete together against other robot teams. Each robot team must have one leader robot, which can be complemented by up to four normal robots and any number of droids.

As in any rumble, the goal is to reduce the energy of enemy robots to 0. Team robots will compete on a battlefield until only one team is left. A standard robot starts with 100 energy and is equipped with a radar and a gun turret. Each time the gun is fired, the robot loses energy according to the payload. If the bullet does not hit any target, the energy is lost. If the bullet hits another robot, energy is transferred from the hit robot to the firing robot.

Apart from the standard robots, there are two special types of robots. The leader robot has an additional 100 energy compared the normal robot. Droids have an additional 20 energy, but are not equipped with radars.

At the end of Sprint 1, your project group will have established a relationship with a robot supplier by pre-ordering either a leader robot, a normal robot, or a droid. The supplier will provide you with a final robot at the end of Sprint 3, but your group will have a chance to influence the evolution of the robot during Sprints 2 and 3. The project manager in your group is responsible for developing a strategy around the robot you have pre-ordered – to guide your feedback, you will receive two prototypes for evaluation: an

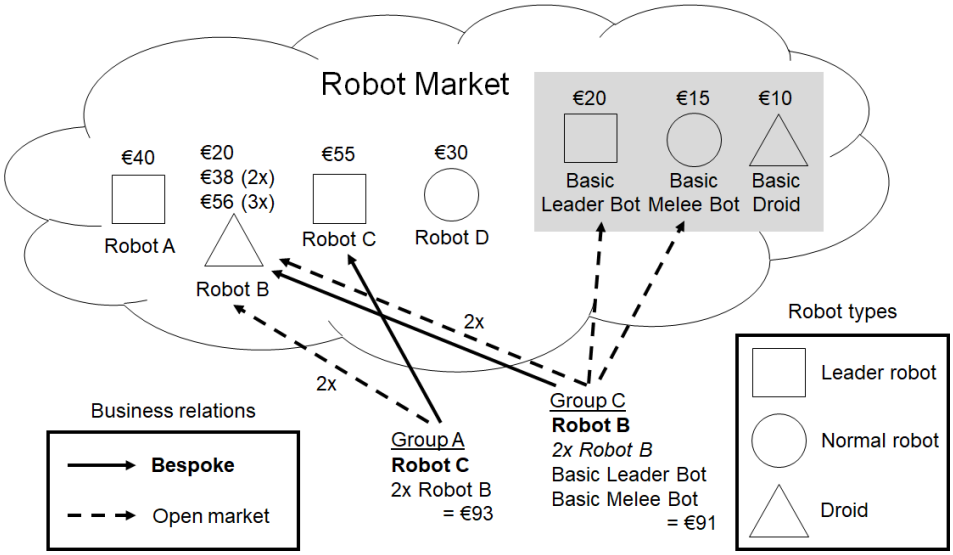

Figure 2.5: Example of two project groups purchasing robots for their teams.

Alpha release after Sprint 1, and a Beta release after Sprint 2.

At the end of Sprint 3, you will complement the robot you have ordered with other robots from the Robot Market. The goal of your strategizing is to identify and purchase additional robots that can successfully cooperate with the robot from your supplier. It is important to keep an eye on the Robot Market during Sprints 2 and 3, both to find the best robots to purchase, and to understand what type of features you should provide through your own robot under development. Note that three robots are always available at fixed prices: 1) Basic Leader Bot €20, 2) Basic Bot €15, and 3) Basic Droid €10.

Each robot team must adhere to five strict constraints:

1. Each team must have one leader robot.
2. Each team must at least have three robots.
3. Each team must not have more than four normal robots.
4. Each team must not have more than ten robots (a Robocode constraint).
5. Each team must not exceed a total cost of more than €100 worth of robots.

After the acceptance testing, concluding with acceptance test reports and potential business claims at L7, the regulatory body will convene and decide if any bespoke customer should pay less than the total price initially agreed upon. Once all business claims have been settled, all groups will know if they receive any money back – remember that even if no final robot is delivered, not more than 75% of the initial price will be returned. Soon after the claims have been settled, the project managers of all student groups shall hand in their final signed purchase order in the department mailbox at L8. Note that

all groups must also immediately request the supplier of their leader robot which color scheme it should communicate to the rest of the team. Note that there is not much time between L8 and the LU Rumble – this must be given the highest priority for the engineering department! Furthermore, if a team purchases the Basic Leader Bot, the requested color scheme must be requested from the course responsible.

3 Project phases

3.1 Project inception

The first week of the project is referred to as the project inception. The project inception phase is dominated by project group development, i.e., forming groups, assigning roles, and aligning goals and expectations.

According to a well-known model of group development, proposed by Tuckman already in 1965, four distinct phases are all necessary and inevitable for a group of people to face up to challenges, collaborate successfully, and to deliver results. Being aware of the four phases might accelerate the less productive phases and mitigate group conflicts:

1. **Forming** The group meets and learns about the project and the required activities. In the beginning, individuals tend to behave quite independently – all trying to comprehend what is about to happen. Individuals might be motivated and on their best behavior, but typically the understanding and expectations vary considerably.

   In the project, groups will be established already during the break in the first lecture, i.e., before lunch the very first Monday. The individual group members are expected to read the project instructions carefully as early as possible.

2. **Storming** After a while, some disagreements and personality clashes are likely to emerge within the group. People are people, and people are different. Tolerance of each group member should be emphasized, otherwise the group will fail or rely on individual "hero efforts". This phase can become destructive to the group, and some teams will never develop past this stage. On the other hand, settling disagreements within the team can make members stronger, more versatile, and able to work more effectively together. Note that some groups never enter this stage at all, i.e., transitions directly from forming to norming are possible.

   In the project, the supervisors and the regulatory body will be particularly responsive to questions and clarifications during the first week. Also, the group communication will be kick-started by two physical meetings soon after the formation, i.e., exercise sessions on Wednesday and Thursday the first week.

3. **Norming** Resolved disagreements and personality clashes lead to a spirit of co-operation – this typically happens when the group becomes aware of the external competition and share a common goal. In this stage, all group members take the responsibility and agree to work for the success of the common goal. The group members accept others as they are and make an effort to move on.

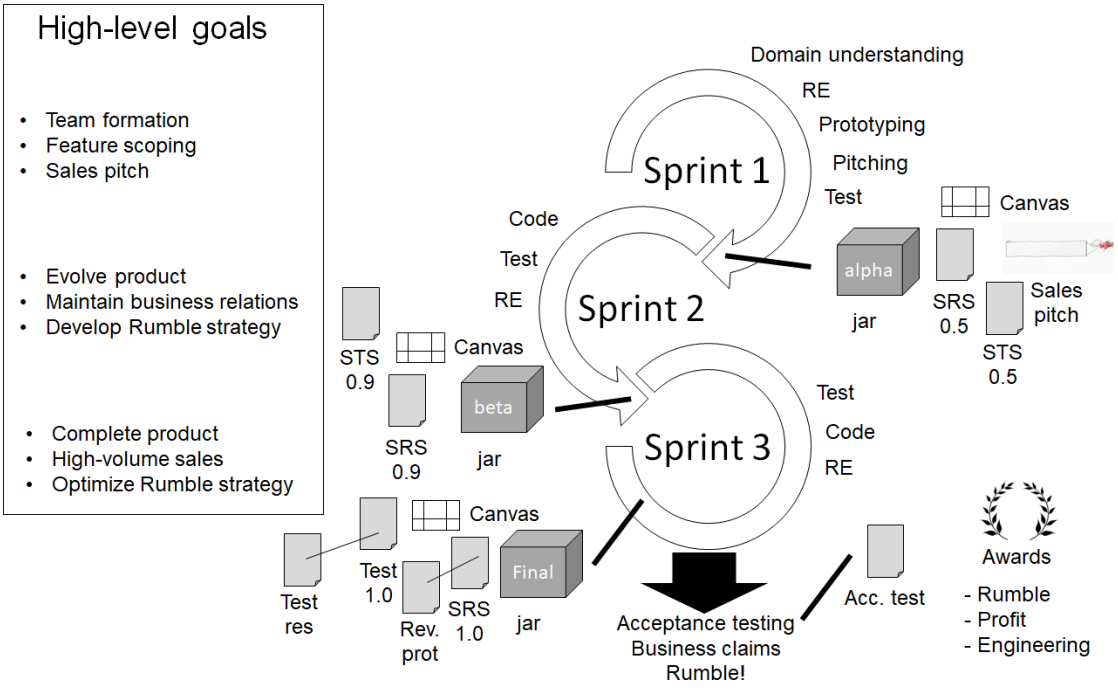

Figure 3.1: The ETSA03 process model, organized in three sprints.

In the project, the first week concludes with assigning roles within the group, see Section 4.1. All members of the group will also sign a collaboration contract, specifying the roles, and the intentions to collaborate successfully. The contracts will be collected in the department mailbox.

4. **Performing** When the group norms and roles have been established, focus is shifted to achieving common goals. However, groups might revert to earlier circumstances in certain circumstances, e.g., unexpected events, emerging personal conflicts, or changing leadership. In the project, the goal is of course to remain in the performing stage until the LU Rumble!

3.2 Process model

The process model describing the product development encompasses three sprints, each with a separate set of deliverables.

3.2.1 Sprint 1

The first sprint overlaps with the project inception as the groups will start working on robot development already at the first exercise sessions in the first week. The high-level goals of Sprint 1 are to successfully do: 1) feature scoping, 2) robot prototyping, and 3) developing a sales pitch. This means that the project group will be focused on

engineering and monetizing during Sprint 1. Some strategizing will also occur, as the project manager will purchase a robot from a supplier after the Robot Fair.

The first steps of the requirements engineering involves analyzing what type of robot would be in high demand by the other project groups. This type of market analysis is no easy task to conduct, and surely the requirements engineer will discuss intensively with the project manager and the sales engineer. Engineering a software product means making a series of successful technology and business decisions, and already during Sprint 1 the project group needs to set the direction for the engineering department. High-level requirements are referred to as features, and the task of deciding which features to implement in the product is called **feature scoping**. At the end of Sprint 1, SRS v0.5 should specify the features the group intends to implement in their robot. The features should be promising enough to generate good offers by other project groups, but do not promise too much (aka. "over scoping") – if you do not deliver a robot that fulfils the specification in the final release, your customer will most likely file business claims to the regulatory body! On the other hand, the SRS v0.5 is not carved in stone. There will be negotiations during Sprints 2 and 3, and if both the customer and the regulatory body accept your change requests, you can both modify and remove features later – and adding new features is of course always possible.

A central part of the engineering in Sprint 1 is the **robot prototyping**, an activity that serves two main purposes. First, it helps your project group experiment with different candidate features for the robot under development. There is no better way to learn the Robocode domain than by practically implementing different approaches and evaluating them. Second, the prototyping will lead to an Alpha version of the Robot. The customer will obtain the Alpha release, which will enable much richer feedback than an SRS alone. To ensure that testing is part of the development process from the start, Sprint 1 will introduce unit testing in the second week.

The marketing concept is the strategy that an organization implements to satisfy customers needs, increase sales, maximize profit, and beat the competition. The customers' focus and values are the routes to achieve sales and maximize profits. In the project, the marketing concept is a customer-centered "sense and responds" philosophy. Your project group's job is not to find the right customers for your robot, but rather to find the right robot for your customers – first for the long-lasting supplier-customer relationship, and later through the Robot Market. At the end of Sprint 1, the course will organize a Robot Fair. Each group will get a 2 minute slot to **pitch** their product through a video. All project groups will also have a folder to showcase promotion material on a "Display window" Google Drive folder shared with all groups.

Table 3.1 shows the resulting deliverables from Sprint 1, their receivers, the responsible roles, and when they are due.

Table 3.1: Deliverables due after Sprint 1, i.e., in relation to the Alpha Release. But why is the sales engineer responsible for the dource code comments!? This is primarily to balance the work effort in the project, but it also ensures that the sales engineer is somewhat connected to the source code of the project.

Deliverable	**Receiver**	**Responsible role**	**When?**
Lean canvas (latest version)	Supervisor	Project manager	L3
SRS v0.5	Customer+ Supervisor	Requirements engineer	L3
Robot Alpha version (jar)	Customer+ Supervisor	Software architect	L3
Class diagram	Supervisor	Software architect	L3
Javadoc	Supervisor	Sales engineer	L3
STS v0.5	Customer+ Supervisor	Quality engineer	L3
Sales pitch + promotion folder	Course responsible	Sales engineer	L2
Purchase array	Dept. mailbox	Project manager	L4

To support Sprint 1, the following sessions are scheduled:

- Lecture 1: Robocode, Requirements engineering
- Lab 1: Robot development in Eclipse
- Exercise 1: Product planning
- Lecture 2: Unit testing, git, object-oriented design
- Lab 2: Refactoring and JUnit
- Exercise 2: Marketing communications

3.2.2 Sprint 2

In Sprint 2, the project group should be up to speed both regarding engineering and monetizing. Now the monetizing will decrease and the strategizing will increase in its place. The high-level goals of Sprint 2 is to: 1) evolve the robot, 2) maintain the customer relationship, and 3) develop a strategy for the LU Rumble.

In Sprint 2, you will have an established customer who has received your Robot Alpha release. As that project group will be keen to win the LU Rumble, you will likely get feedback that they believe will guide your development to a successful robot. During the remainder of the project, maintaining a healthy **customer relationship** is critical. Your sales engineer is the main point of contact with the customer.

Table 3.2: Deliverables due after Sprint 2, i.e., in relation to the Beta Release.

Deliverable	**Receiver**	**Responsible role**	**When?**
Lean canvas (latest version)	Supervisor	Project manager	L5
SRS v0.9	Customer+ Supervisor	Requirements engineer	L5
Robot Beta version (MVP)	Customer+ Supervisor	Software architect	L5
Class diagram	Supervisor	Software architect	L5
Javadoc	Supervisor	Sales engineer	L5
STS v0.9	Customer+ Supervisor	Quality engineer	L5

Based on your customer's feedback, your own vision – and possibly some negotiation in between – you will **evolve the robot** during Sprint 2. The features will be broken down into detailed requirements, and the STS will be elaborated to also cover system testing. At the end of Sprint 2, you will deliver Robot Beta release to your customer, constituting an MVP, i.e., a minimal product that still brings some value. To be in a good position for the next sprint, the project group should try to let the lean canvas, SRS, source code, STS, and marketing concept co-evolve – otherwise they might diverge considerably, increasing the risk of a dissatisfied customer, wasted effort within your group, and reduced sales on the Robot Market. Make sure that any changes you make to the SRS are traceable, i.e., make sure there is a "paper trail", in case the regulatory body needs to settle business claims from your customer at a later stage. Significant changes to the SRS must be done in agreement with the customer and documented. If the supplier organization does not store a chain of evidence, the regulatory body is likely to side with the customer in legal contentions.

Finally, you will develop a first draft of your **Rumble strategy**. The cornerstone of your robot team will be the robot you pre-ordered. Your project manager will be responsible for communication with your supplier's sales engineer. Just like your project group should be responsive to your customer, your supplier should respond to your requests.

Table 3.2 shows the resulting deliverables from Sprint 2, their receivers, the responsible roles, and when they are due.

To support Sprint 2, the following sessions are scheduled:

- Lecture 3: Black-box testing, software architecture
- Lab 3: Automated system testing with RobotTestBed
- Exercise 3: Requirements specification
- Lecture 4: Project planning, document reviews

3.2.3 Sprint 3

In Sprint 3, the project group shall conclude and tie up all activities. The engineering should lead to a successful robot, the monetizing should ensure that it is well-received on the market, and the strategizing shall finalize the robot team shall be competitive in the upcoming LU Rumble. The high-level goals of Sprint 3 are to: 1) complete the product, 2) generate high-volume sales, and to 3) optimize the Rumble strategy.

The engineering department will be busy implementing the remaining requirements and putting the finishing touches to the robot. This means **completing the product** and preparing the Robot Final release. When the Final Robot is released, it is important that the lean canvas, SRS, STS, and the marketing concept are all aligned. Your customer will perform acceptance testing of your robot – rest assured that they will scrutinize your deliverables to identify any discrepancies. The customer will be greatly motivated to assure that the Robot Final release complies with the SRS – their entire robot team for the LU Rumble might rely on it! If the customer is not satisfied, the regulatory body is likely to receive a business claim based on their acceptance testing. However, your project group will provide the regulatory body with evidence that everything in the SRS has been verified by providing a complete test report based on STS v1.0.

The sales department will keep active communications with the customer, to make sure the final engineering efforts adhere to their expectations. Close communication with the customer will reduce the risks of incoming business claims. However, the sales engineer will also focus considerable effort on the robot offering on Robot Market – trying to make the robot as attractive as possible on the open market, i.e., the sales engineer will maintain the content in the "Display Window" on Google Drive with the intention to **generate high-volume sales**.

The executive department will keep a close eye on the engineering efforts of the supplier. After all, this is the supplier's last chance to deliver the robot that has been specified in the SRS. At the same time, your project manager will analyze the Robot Market to make a final decision of your robot team composition, i.e., **optimizing the Rumble strategy**.

Table 3.3 shows the resulting deliverables from Sprint 3, their receivers, the responsible roles, and when they are due.

To support Sprint 3, the following sessions are scheduled:

- Lecture 5: Development processes
- Lab 4: Code coverage, static code analysis, Javadoc
- Exercise 4: Individual SRS review, review meeting

3.3 Post release

The phase following the process model is referred to as post mortem. In this phase, your project group will perform acceptance testing of your supplier's Robot Final release. If

Table 3.3: Deliverables due after Sprint 3, i.e., in relation to the Final Release.

Deliverable	**Receiver**	**Responsible role**	**When?**
Lean canvas (final version)	Supervisor	Project manager	L6
SRS review protocol	Supervisor	Requirements engineer	L6
SRS v1.0	Customer+ Supervisor	Requirements engineer	L6
Robot Final version	Customer+ Supervisor	Software architect	L6
Class diagram	Supervisor	Software architect	L6
Javadoc	Supervisor	Sales engineer	L6
STS v1.0	Customer+ Supervisor	Quality engineer	L6

Table 3.4: Deliverables due after Sprint 3, i.e., the Final Release.

Deliverable	**Receiver**	**Responsible role**	**When?**
Fixed Robot Market prices	Robot Market	Sales engineer	L7
Acceptance test report	Supervisor	Project manager	L7
Business claims (optional)	Course responsible	Project manager	L7
Purchase orders and team composition	Course responsible	Project manager	L8

the acceptance testing identifies flaws, you will file business claims to the regulatory body – possibly deducting from the total payment. Furthermore, after settling the business claims, the project manager will submit the final purchase orders to the regulatory body, i.e., the robot team composition for the LU Rumble will be finalized.

Table 3.4 shows the resulting deliverables from the post release phase, their receivers, the responsible roles, and when they are due.

To support the post release phase, one session is scheduled:

- Lecture 6: Configuration management, software product lines

3.4 LU Rumble

The final phase consists of one single event: the LU Rumble. The LU Rumble will take place during the final lecture of the course. LU Rumble will follow a cup pairing according to single elimination knockout rules, see Figure 3.2. All robot teams will enter the bracket at random. LU Rumble will also have a third place playoff match. Each match in the LU Rumble will be played in a best-of-five format, and displayed live at reasonable speed for the entertainment of the spectators. Standard rules for Robocode team matches will apply on a 1200x1200 battlefield.

Immediately after the LU Rumble, three awards will be given:

Figure 3.2: The LU Rumble bracket. In addition, there will be a third place playoff match.

- The Strategizer Award goes to the student group that won the LU Rumble.
- The Monetizer Award goes to the most profitable student group. The profit is calculated as income from robot sales plus any money left from the initial €100 budget.
- The Engineering Award goes to the most valuable robot during LU Rumble. The robot's value will be calculated as the total score of all robot instances during the LU Rumble, divided by the (non-zero) number of robot appearances (each round a robot instance enters the battle is counter as one appearance).

4 Detailed instructions

This section presents some aspects of the project in detail: 1) the roles you should assign within the project group, 2) the time reporting, and 3) the course infrastructure.

4.1 Role assignment

During the course, all students are expected to contribute to all activities. The purpose of the roles is not to isolate work tasks that could be individually completed. Software engineering is a collaborative effort, and completing work tasks in isolation is not part of a scalable methodology. Instead, we stress the value of "collective ownership" during the project, not only for the source code as described in Section 2.1.2, but for all artifacts produced. To ensure that all group members can contribute to the technical parts of the project, the lab sessions are mandatory – to further highlight this, all group members are assigned the roles of developer and tester. However, all group members will also be assigned a third role.

The purpose of assigning roles in the project is to comply with the ETSA03 course description regarding "judgment and approach", i.e., understanding that software engineering is a complex endeavor involving several different roles, and that working as a software engineer might mean shouldering rather different roles in an organization. Second, all roles with be responsible for some deliverables. The regulatory body will use the group member with the corresponding role as the primary point of contact in case any deliverable is missing or critically incomplete.

As the project progresses through phases and sprints, different roles with have different responsibilities. The project group will track the individual efforts through weekly time reporting, submitted by the project manager to the regulatory body – this enables identification of major deviations. The reported figures should be complemented by a qualitative analysis of the causes, and the project manager is responsible for balancing the workload to make sure the contributions are comparable, see 4.2 for further information. Below are the different roles, ordered alphabetically, followed by the deliverables they are responsible for.

Configuration manager Manages the version control of source code and documentation. Manages the the build and release processes. Ensures that all developers can access the source code repository, build and test the software locally, and commit changes. Responsible for packaging and delivering the Alpha, Beta, and Final releases. Keeps an eye on the CI server to ensure the builds are passing. Identify a service minded member willing to become your git superuser and help others.
Deliverables: Alfa release, Beta release, Final release

Project manager Maintains a holistic picture of the project, should always be aware of where the current risks and bottlenecks are. Coordinates the group, ensures smooth communication, and keeps an eye on the coming deadlines. Furthermore, responsible for purchasing other robots. Main contact point for the regulatory body. Identify a group member with leadership skills that likes to maintain a sense of control.
Deliverables: Weekly time reports, purchase array, acceptance test report (and potential business claims)

Quality engineer Overall responsibility for testing the software. Develops a test strategy and documents the approach to unit testing and system testing in the Software Test Specification. Acts as a test lead among the testers to ensure that automated test cases are developed according to plan – also responsible for the automated system test cases. The role is suited for a person with both strong development skills and careful technical writing.
Deliverables: STS v0.5, STS v0.9, STS v1.0, automated system test cases

Requirements engineer Needs to focus on the end-user perspective and is responsible for feature scoping and specification of detailed requirements. A requirements engineer is a generalist that bridges engineering and business. Requirements should be crystal clear – assigning your "language police" to this role might be wise.
Deliverables Lean canvas, SRS v0.5, SRS v0.9, SRS v1.0

Sales engineer Responsible for marketing of the robot, the initial sales pitch, customer communication, and negotiations. During development, the sales engineer will also act as a proxy-customer if needed, i.e., steering development decisions in the interest of the real customer. Toward the end of the development project, the sales engineer is responsible for the concluding system testing. Identify your natural salesman, a person with excellent communication skills.
Deliverables: Sales pitch video, Javadoc, test report.

Software architect Overall responsibility for the design and implementation of the software. Act as a development lead among the developers. Assign a group member comfortable with the art and science of programming, as well as an interest in object-oriented design. Responsible for source code and unit test cases.
Deliverables: `jar_v0.5`, `jar_v0.9`, `jar_v1.0`

4.2 Time reporting

All groups shall report how much effort they spend during the second phase of the project, i.e., the three development sprints. The project manager is responsible for collecting how many hours each member of the team works on the project on a weekly basis. In the end of each week, the regulatory body will collect the information and present an overview during the Monday lectures. Note that the reported hours shall **not** include the scheduled time for lectures, computer labs, or exercises – even if they are

(Estimated effort in hours)	Week 14	Week 15	Week 16	Week 17	Week 18	Week 19	*Effort per role*	
Project manager	4	25	24	12	15	10	*90*	
Requirements engineer	3	8	25	10	20	13	*79*	
Software architect	6	7	9	14	40	30	*106*	
Quality engineer	5	7	5	10	30	10	*67*	
Configuration manager	3	10	6	14	15	25	*73*	
Sales engineer	3	30	7	9	17	8	*74*	
Effort per week	*24*	*87*	*76*	*69*	*137*	*96*	489	
							489	**Total project effort**
							480	**Project budget**
							-9	**Difference**

Figure 4.1: Example of time reporting table.

directly connected to work in the projects. This rule is based on the 4 ECTS that are provided for completing the project, whereas the remaining 2 ECTS of the course cover the lab sessions.

Project managers report hours in the dedicated spreadsheet on Google Drive. Each group has a separate tab in the document, as presented in Figure 4.1. All numbers shall be entered in the white cells. After each development sprint, it is a good practice to do a sprint retrospective, i.e., reflect on how the group worked during the sprint and what could be improved. Typical questions that can drive a retrospective discussion are:

- What went well in the Sprint? This question might identify what the group should **continue doing** in the next sprint.
- What could be improved? Based on this question, you might identify things the group should **start doing** and/or things the group should **stop doing**.
- Did the group **balance the work load** among the roles reasonably well? Do any of the roles need to delegate tasks to others?

4.3 Software Releases: Alpha, Beta, and Final

The robot shall be released after each development sprint, each time constituting a more mature software product. In ETSA03, you shall create each software release in two variants: 1) a customer release and 2) a regulatory body release. Groups will upload their releases in designated folders on Google Drive.

The **customer release** shall contain the latest versions of the following:

- SRS
- Robot packaged as a jar-file – without source code!
- STS

- Optional: Release notes – if you want to clarify anything related to the release. Simply create a text file called `ReleaseNotes_vX.txt` and replace X with 0_5, 0_9, or 1_0.

The **regulatory body release** shall contain everything in the customer release, but also the latest version of the following artifacts:

- Lean canvas
- Robot packages as a jar-file – including source files!
- Class diagram (excluding test code)
- Javadoc (excluding test code)
- A review protocol from the review meeting at Exercise 4 (only for the Beta release)

5 Expectations per deliverable

This section briefly explains what is expected of the various deliverables.

Collaboration contract The first deliverable is a contract signed by all group members, stating agreed upon collaboration rules and the role assignment within the group.

Sales pitch A video pitch to be played at the Robot Fair, not longer than 2 min. The video shall be recorded in a standard video format, but you are free to choose the structure and content. For example, you can choose whether to record yourself speaking, a screencast, or a mix of both. The sales pitch shall communicate your message, you are not supposed to complement it during the Robot Fair with live talking. You are of course not allowed to include any offensive or illegal content in the marketing communications. You will also get a shared folder on Google Drive which you can use freely to promote your sales – be creative!

SRS The software requirements specification shall be delivered in three versions. Each version should be a more mature specification of the robot under development.

- **v0.5** A skeleton of an SRS adhering to the provided template. The main features that will be implemented shall be listed.
- **v0.9** The features shall be broken down into detailed requirements.
- **v1.0** A final SRS that provides a complete specification of the robot.

SRS review protocol The review meeting at Exercise 4 will result in a formal review protocol according to the provided template. The protocol shall report all issues that were identified during individual reviews, and also estimate the number of remaining issues based on the capture-recapture method.

The robot The robot shall be released in three versions: Alpha, Beta, and Final. The software architect shall each time package the robot as jar-files using the Robot Packager in Robocode, in two different versions per release: 1) without source files for the customer, and 2) including source files for the supervisor. The source code shall be maintainable, e.g., well-structured, commented, and using suitable variable names.

- **Alpha** An early prototype of the robot. No expectations regarding features, but the customer shall be able to field it in a Robocode battle.
- **Beta** A minimum viable product (MVP), i.e., a robot with just enough features to satisfy your customers. The source code classes shall at least partly be tested by a suite of unit test cases. No automated system test cases are required for the Beta release.

Final A polished robot, ready for LU Rumble. The source code classes shall mostly be tested by corresponding unit test cases. At least some system test cases shall be automated using JUnit and RobotTestBed.

Class diagram The object-oriented design shall be documented in a UML class diagram. You can use any tool to create the diagram, either a tailored UML tool or a standard drawing tool – you can even scan a pen and paper sketch or take a picture of a whiteboard drawing.

Javadoc The robot source code shall be documented and packaged using the standard Javadoc tool.

STS The document shall describe how the group uses testing to increase the quality of the robot, i.e., what measures are taken to decrease the risk of releasing a product with bugs. Furthermore, system test cases used to verify that the SRS is completed shall be specified.

v0.5 An initial STS skeleton adhering to the provided template.

v0.9 A fairly complete draft of the STS, partly specifying unit testing and system testing.

v1.0 A final STS that specifies a trustworthy approach to verify the SRS.

Test report The final release shall be accompanied by a test report that shows the result of executing the STS. The test report shall follow the structure in the STS Appendix.

Purchase array The purchase array shall specify valid offers for all other groups' robots and be signed by the project manager.

Robot Market prices The groups shall specify the price of their robot on Robot Market, and also any bundle deals.

Acceptance test report When the final release of another group's robot has been received, you shall test it to see if it complies with its SRS. You shall write an acceptance test report clearly stating how you tested the robot and what the results were. You are free to choose any format for this deliverable.

Business claims This is an optional deliverable that you can send to the regulatory body. If you are not satisfied with the final release you received from another group, you can argue for your case in a written business claim. You are free to choose any format for the deliverable.