NTNU – Trondheim
Norwegian University of
Science and Technology

CESAR - text vs. boilerplates

What is more effcient - requirements written
as free text or using boilerplates (templates)?

Vegard Johannessen

Master of Science in Computer Science
Submission date: June 2012
Supervisor: Tor Stålhane, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem description

Poor requirements specifications are identified by several studies as one of the most important reasons for cancellation, delay or cost overrun of software projects.

The most common method of expressing requirements is to use free text. This method is seen as flexible and easy to use, but offers little support for ensuring the quality of the requirements.

Boilerplates have been developed as an alternative to free text. This project will study some important differences between the two methods, in order to determine the value of using boilerplates, as well as to discover any problems connected with introducing the method to somebody that is unfamiliar with it.

Abstract

This thesis examines the differences between boilerplates and free text as methods to write software requirements. Through an experiment, the methods will be tested from both the requirements engineering standpoint, and from the standpoint of a stakeholder looking to understand the requirements.

Preface

Boilerplates are a semi-formal method for writing requirements. This report examines the differences between using boilerplates and natural language, or free text.

The thesis is the first study in this area, and provides insight into the strenghts and weaknesses of boilerplates, as well as suggesting which areas need more research.

The experiment performed during the project was carried out using students from the second grade of the Computer Science program from NTNU, and was held in an auditorium. Results were ambiguos. It is the candidate's opinion that these two factors influenced the results negatively. In order to get clearer answers about the quality of boilerplates as a requirements engineering method, it is neccessary to do a more real-life experiment, using more experienced test subject, and also using tools like DODT to write the requirements.

Preface

Boilerplates er en semi-formell metode for å skrive krav. Denne rapporten undersøker forskjellene mellom å bruke boilerplates og naturlig språk eller fri tekst.

Masteroppgaven er den første store forskningen på dette området, og byr på innsikt i styrker og svakheter ved boilerplates, i tillegg til å foreslå hvilke områder som bør forskes videre på.

Eksperimentet som er utført som del av prosjektet ble gjennomført med studenter fra andre året ved Datateknikkstudiet ved NTNU, og ble holdt i et auditorium. Resultatene var tvetydige. Det er kandidatens mening at disse to faktorene påvirket resultatene på en negativ måte. For å få klarere svar på kvaliteten av boilerplates som kravspesifiseringsmetode, så er det nødvendig å gjennomføre et eksperiment som er mer virkelighetsnært, ved å bruke mer erfarne testpersoner og å bruke verktøy som DODT til å skrive kravene.

Acknowledgments

I would like to thank my supervisor, Professor Tor Stålhane, for his support, guidance and for being an invaluable discussion partner through this project.

Vegard Johannessen
Trondheim, June 12, 2012

Contents

List of Figures

List of Tables

Glossary

atomic requirements. The requirement does not contain conjunctions [?]. xvi

boilerplate A template consisting of fixed syntax elements and placeholders for attributes. ix, xiii, 1–5, 7, 9–13

boilerplate requirement Requirements written using boilerplates, with loose limitations on form or vocabulary. 16–18, 20, 22

cohesive about requirements. The requirements address one and only one thing [?]. xvi

complete about requirements. The requirement is fully stated in one place with no missing information [?]. xvi

consistent requirements. The requirement does not contradict any other requirement [?]. xvi

current requirements. The requirement has not been made obsolete by the passage of time [?]. xvi

DODT requirements engineering tool for writing requirements using boilerplates.. 36

feasable requirements. The requirement can be implemented within the constraints of the project [?]. xvi

free text requirement Requirements written in a natural language, like English, with loose limitations on form or vocabulary. ix, 2–4, 7, 13, 16–20, 22

null hypothesis “states that there are no real underlying trends or patterns in the
experiment setting; the only reasons for differences in our observations are coincidental. This is the hypothesis that the experimenter wants to reject

with as high significance as possible.” [?] In this project the null hypothesis is mostly used as the assumption that there is no difference between two treatments. 17

quality of requirements. Here: a requirement that is cohesive, complete, consistent, atomic, unambiguous and verifiable. Other characteristics, which are not considered in the frames of this project, are traceable, feasable and that the requirement is current and has a specified importance. These characteristics does not relate directly to the formulation of the requirements, and are thus not relevant for this topic.. i

specified importance of requirements. Many requirements represent a stakeholder-defined characteristic; the absence of which will result in a major or even fatal deficiency. Others represent features that may be implemented if time and budget permits. The requirement must specify a level of importance [?]. xvi

traceable requirements. The requirement meets all or part of a business need stated by stakeholders and is authoritatively documented [?]. xvi

unambiguous requirements. The requirement is concisely stated without recourse to technical jargon, acronyms (unless defined elsewhere in the Requirements document), or other esoteric verbiage. It expresses objective facts, not subjective opinions. It is subject to one and only one interpretation. Vague subjects, adjectives, prepositions, verbs and subjective phrases are avoided. Negative statments and compound statements are avoided [?]. xvi

verifiable requirements. The implementation of the requirement can be determined through one of four possible methods: inspection, demonstration, test or analysis [?]. xvi

Chapter 1

Introduction and Overview

1.1 Background and Motivation

Research shows that poor requirements are at the top of the list of reasons for why software projects fail or are challenged (by cost- or time overruns, or providing fewer
features or functions than originally specified). The Standish Group [1995] places incomplete and changing requirements and specifications second and third on the list of "Project Challenged Factors", whereas incomplete requirements are the number one reason for why projects are cancelled.

This makes it clear that the state of software requirements specifications is generally not good enough. The Standish Group does not specifiy the reasons behind the numbers, but incomplete and changing requirements suggests either a poor requirements engineering process or misunderstandings about the agreed upon requirements. This report will focus on the latter, looking at whether a semi-formal languagesuch as boilerplates will make requirements easier to understand.

1.2 Goals and Research Questions

Main goal

To determine whether boilerplates are a better method than free text for writing software requirements.

During the specialization project, one of the uncertainties was how to make a definitive statement about the quality of the requirements. We decided that such questions needed to be answered by both an objective and a subjective evaluation. An objective evaluation will be carried out by looking at the results of the experiment performed, more specifically by the quantitative data from the tasks (number of errors found, the percentage of functionality extracted from the requirements, etc.)

The success or failure of this goal will be determined by the results of the research questions below, which in turn will be answered by quantifiable data from the experiment. The research questions will be used to formulate hypotheses, and the questions and tasks presented in the experiment will each be based on one or more of the hypotheses..

Research question 1

Is it easier to prevent ambiguities when writing boilerplate requirements than free text requirements?

Based on the work done in the specialization project, one of the factors that was decided to be used when assessing the quality of the methods, is the amount of ambiguities in the requirements. The Standish Group [1995] names a clear statement of the requirements as the third most important factor for project success. How good the method is in preventing ambiguous requirements should therefore be one of the measures of quality. This factor is looked at from two perspectives, the first of which is how ambiguous the requirements are when first written. To answer this question, it is necessary to analyze requirements written with both methods, to se which method produces the most ambiguous requirements.

Research question 2

Is it easier to discover ambiguities in boilerplate requirements than in free text requirements?

The other perspective that will be used when looking at ambiguities, is how easily they are discovered when reading the requirements specification. Theoretically, boilerplates such as BP21 and BP22 (Table 2.3) should make the requirements more accurate, but it is necessary to test whether users experience them as such.

Research question 3

> Is it easier to prevent inconsistencies when writing boilerplate requirements than free text requirements?

Inconsistency is the other factor it was decided to look at when judging the quality of requirements. Inconsistent requirements will either lead to faulty or incorrect software, or changes in the requirements specification later in the project, which is named as one of the most important reasons for challenged or cancelled projects. This factor will also be measured from two perspectives; the number of inconsistencies that are introduced when writing requirements and how many inconsistencies are discovered when reading a requirements specification.

Research question 4

> Is it easier to discover inconsistencies in boilerplate requirements than free text requirements?

As with ambiguities, the experiment will also study whether the test subjects find more inconsistencies in boilerplate requirements than in free text requirements. This perspective is used in order to see whether the use of templates and ontologies will make it more apparent that two requirements are conflicting.

Research question 5

> How much introduction to boilerplates is necessary to both understand and be able to write new requirements using boilerplates?

One of the major findings of the specialization project was the need for proper training in order to understand the concept of boilerplates. The test subjects who had only a written introduction eventually gave up on most of the tasks presented to them, while those who also got a ten minute introduction to both boilerplates and the cases used, turned out to have far more success. Still, they used a rather limited set of different boilerplates. Some more testing is therefore required to see how much training is necessary before the subjects will be able to do the tasks.

1.3 Project Scope

This thesis will look at the differences between free text requirements and boilerplates. It will not discuss the effect of boilerplates for development methods, such as iterative (Agile or Scrum) vs sequential development models (Waterfall model), or differences between smaller and larger projects. The experiment will use only students as test subjects.

Further research is therefore necessary to determine whether the results are valid in all contexts, but they should at least give an indication of the strenght of boilerplates vs FTRs.

1.4 Research Method

In order to answer the research questions listed in section 1.2, it is necessary to perform one or more experiments. Wohlin et al. [1999] presents four research methods for use in software engineering context:

The scientific method	The world is observed and a model is built based on the observation, for example, a simulation model.
The engineering method	The current solutions are studied and changes are proposed, and then evaluated.
The empirical method	A model is proposed and evaluated through empirical studies, for example, case studies or experiments.
The analytical method	A formal theory is proposed and then compared with empirical observations.

Table 1.1: Research methods. Adapted from Wohlin et al. [1999].

Of these, the empirical and the analytical method are both useful choices, as they examine the requirements engineering methods in a controlled environment and provides quantifiable results. The empirical method will be chosen for this project. More specifically, the project will be based on a set of experiments, in which both requirements engineering methods are examined. This makes it easier to control that the conditions are the same for both methods, for example that the experience gathered from working with either method first, does not influence the results of working with the other method afterwards (see section 3.5), and also makes it possible to evaluate whether subjects with different previous experience produce different results.

The experiments will be similar to the ones used in the specialization project, but also implementing changes to reflect the feedback and observations gathered from that project. Most noticable, the cases used in the specialization project is replaced. The feedback suggested that the cases using an automatic cruise controller and a steam boiler were too complicated or too far from the test subjects field of expertise. A completely new case has therefore been created for this study.

The experiment will consist of a pre-experiment questionnaire where the participants will give information about their previous experiences, a main part where they will solve several tasks related to the case, and a post-experiment questionnaire where they will evaluate their experience of how the experiment went, how they experienced working with boilerplate, etc. More details concerning the experiment design can be found in chapter 3.

1.5 Contributions

This project is the first that compares boilerplates as a requirements engineering method against regular free text. As such, this report gives insight into which strengths and weaknesses the method displays in an experiment, and discusses challenges surrounding successful requirements engineering. The report also discusses what other factors are important to look at when evaluating requirements and requirements engineering.

As mentioned, this is the first thesis which explores this topic. As such, the paper also serves as a basis for further research.

1.6 Thesis Structure

This thesis follows the following structure:

The first two chapters are introductory, presenting the knowledge neccessary to understand the topics discussed later. Chapter 1 explains the goals of the project and outlines how it was conducted. Chapter 2 gives an introduction to the methods and theories discussed in the thesis.

The next three chapters consern themselves with the experiment performed. Chapter 3 describes how the experiments was designed, chapter 4 describes how it was performed, while chapter 5 presents the collection and analysis of the data derived from the experiment.

Chapter 6 summarizes the results of the experiment, discusses how they should be understood and whether there are any threats to the validity of the results. Finally it tries to answer the research questions posed in section 1.2.

Chapter 2

Theory and Background

2.1 Software Requirements

Software requirements can be sorted into three categories: formal, semi-formal and informal. Formal requirements are easy to understand for machines, but normally require a lot of training, and also places several restrictions on the requirements engineer. Use Cases are one type of formal requirements.

Informal requirements are a lot less restrictive about form and formulation, but does not provide much in terms of automation. Also, informal requirements are difficult to understand for machines. They are, however, easy to get started with, and can be understood by stakeholders who lack training. This project looks at free text requirements, which is one type of informal requirements. (See section 2.3)

Semi-formal requirements are meant to be the middle-way between formal and informal requirements. They try to combine the ease of use of informal requirement with the ability to automate work processes surrounding maintaining the requirements specification. Another goal the semi-formal requirements try to achieve, is aiding the requirements engineer in writing better requirements. This thesis is mainly based on the semi-formal requirements engineering method called boilerplates. That method is further described in the next section.

2.2 Boilerplates

The purpose of using boilerplates is that they help "in knowing how to express certain kinds of requirement in a consistent language". [Hull et al., 2009]

As shown in Figure 2.1, boilerplates consist of two parts; the actual boilerplate or template, and the values you fill in, hereby refered to as attributes. The boilerplates can be chosen from a standardized set, such as the one developed by CESAR or an in-house developed set for the specific company.

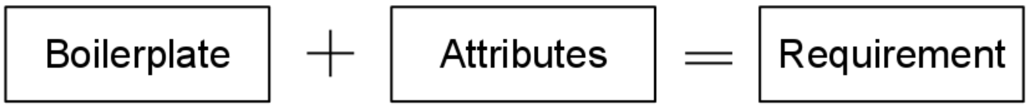


Figure 2.1: Requirements are built by combining a boilerplate and the corresponding attributes

The attributes are then either chosen from a standard repository for the company, or a specific set decided on for each project. An example of the use of boilerplates is provided in Table 2.1. The reason for providing and maintaining a standard attribute repository is to ensure that the same name is given to the same entity for all requirements, and that this wording is consistent both throughout the project and between projects within the organization. This is supposed to prevent ambiguities and misunderstandings that can arise if an entity is given different names, or perhaps is not named properly.

Boilerplate	The <system function> shall provide <system capability> to achieve <goal>.
Attributes	System function: ACC System capability: distance monitoring Goal: a minimum distance to vehicle in front.
Requirement	The ACC shall provide distance monitoring to achieve a minimum distance to vehicle in front.

Table 2.1: Example use of boilerplate

To use the example in Table 2.1, the system capability, *distance monitoring*, could perhaps be refered to as *longitude monitoring* or *a way to measure distance.* Mixing these terms could not only give widely different results, but could also confuse the developer into believing they refered to three different entities.

Zojer et al. [2011] uses the attribute structure shown in Figure 2.2. The *attribute* node on the top is only a parent node, and is not used in itself. Other than that, quantity is used for any measures necessary, regardless of whether it is length, weight, duration or any other measure. It is also important to specify that this can be used in cases where the specific value is not yet determined, in which case it can be instantiated as "TBD", *to be decided.* Also, not filling out the quantity can be used in cases where the values are confidential. If the requirements need to be published, that may then be done without publishing any of the confidential data.

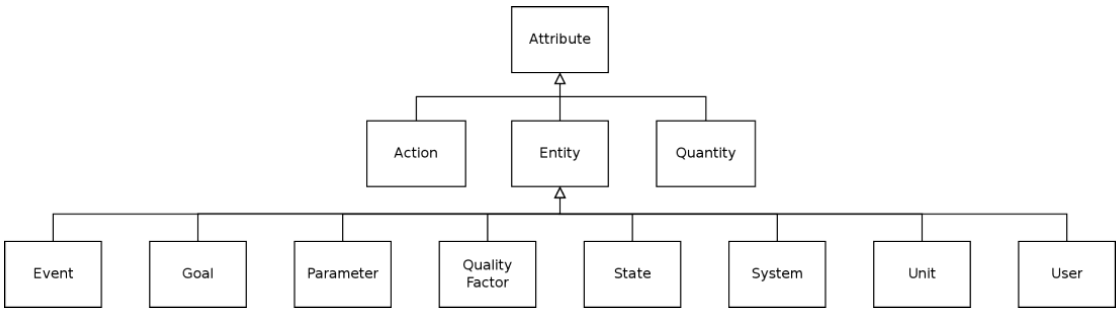


Figure 2.2: Attribute structure (adapted from Zojer et al. [2011])

By using existing boilerplates whenever possible, the requirements will be more uniform, preventing vague or ambiguous requirements. New boilerplates will, perhaps, have to be added as the method matures, but only after it has been decided no existing ones will cover the requirement. As the boilerplates are used over time, the company or organization can gradually achieve uniformity for its requirements, preventing vague or ambiguous requirements leading to costly mistakes.

2.2.1 Boilerplate classifications

Each boilerplate can be classified according to the categories listed in Table 2.2. This helps ordering the requirements. The words in paranthesis refers to the goal type for the boilerplate, such as minimizing or maximizing something.

The following is the complete list of boilerplates as presented in Zojer et al. [2011]. The boilerplates are divided into three types; main- , prefix- and suffix boilerplates. Main boilerplates can stand alone, whereas prefix suffix boilerplates must be prepended or appended, respectively. The two latter are sometimes refered to collectively as *modes*, but will in this report be differentiated.

Classification of boilerplates
Capability
Capacity (Maximise, Exceed)
Rapidity (Minimise, do not exceed)
Mode (while, if, for ...)
Sustainability
Timelines
Operational Constraints
Exception

Table 2.2: Boilerplate Classifications

Main boilerplates

The main boilerplates are standalone, meaning if the attributes are instantiated they can constitute a requirement on their own. For example, BP15, can be instatiated as "<Laptop>shall be able to <reboot>".

BP15: <system> may be <state>
BP16: <system> shall <action>
BP17: <system> shall allow <entity> to be <state>
BP18: <system> shall be <entity>
BP19: <system> shall be able to <action>
BP20: <system> shall have <entity>
BP21: <system> shall have <quality factor> of at least <quantity> <unit>
BP22: <system> shall have <quality factor> of at most <quantity> <unit>
BP23: <system> shall not <action>
BP24: <system> shall not allow <action> BP25: <system> shall not allow <entity> to <action>
BP26: <user> shall be able to <action>

Table 2.3: Main boilerplates

Prefix boilerplates

Prefix boilerplates need to be prepended to one of the 12 main boilerplates, and can then provide conditions like "if <event>" or "while <state>". For example, prepending BP28 to BP19, one could create the following requirement: "If <in sleep mode>, <laptop> shall be able to <resume session>."

BP27: if <event>, ...
BP28: if <state>, ...
BP29: in order to <action> ...
BP30: in order to achieve <goal> ...
BP31: while <state> , ...

Table 2.4: Prefix boilerplates

Suffix boilerplates

Like prefixes, suffix boilerplates needs to be attached to a main boilerplate, like this: "If <in sleep mode>, <laptop> shall be able to <resume session> within <3> <seconds>." (BP 12 attached to BP 28+ BP19)

BP0: ... after <event>
BP1: ... at <entity>
BP2: ... at least <quantity> times per <unit>
BP3: ... before <event>
BP4: ... during <state>
BP5: ... every <quantity> <unit>
BP6: ... except for <action>
BP7: ... for a period of at least <quantity> <unit>
BP8: ... from <entity>
BP9: ... other than <action>
BP10: ... to <entity>
BP11: ... unless <state>
BP12: ... within <quantity> <unit>
BP13: ... within <quantity> <unit> from <event>

Table 2.5: Suffix boilerplates

2.2.2 Benefits to using boilerplates

The general goal of using boilerplates is to standardize the formulation of requirements, both with regards to structure and dictionary, so as to create consistency and prevent ambiguities.

Whether these benefits are real or just assumed is part of what this report will try to answer. Not much research has been published on this field at the present, meaning the benefits listed below must so far be considered theoretical.

Standardized formulation The boilerplates forces requirements to be expressed in a certain way, helping similar requirements look alike. This should help stakeholders (developers, customers, management, etc) understand the requirements better, and also prevent confusion as to what the requirement means. (See also “Prevent disambiguity”)

Prevent disambiguity The rigidness of the boilerplates prevents a vague language, and encourages specifying the exact meaning. For examples, see BP2, BP7, BP21, and BP29 in tables 2.3 - 2.5.

Prevent inconsistencies A clearer language and specification of purpose (BP29, BP30) helps developers and other stakeholders see dependencies between requirements, thus preventing inconsistencies.

Uniformity of language/Consistency In addition to the boilerplates, the attributes should be selected from a repository, which ensures that attributes are named the same throughout the requirements specification, and also between projects. A common understanding of the requirements reduces the possibility of ambiguities and inconsistencies.

Easy to understand Boilerplates are written in what is called a semi-formal language. There are certain restrictions to the vocabular and the structure of the sentences, but they are less strict than, say, a UML diagram. Carew et al. [2005] found that people understand informal representations better than formal ones. The boilerplates, when attributes are initialized, appear to be normal text, and should therefore be easier to understand than formal representations, while still maintaining the benefits of a semi-formal method.

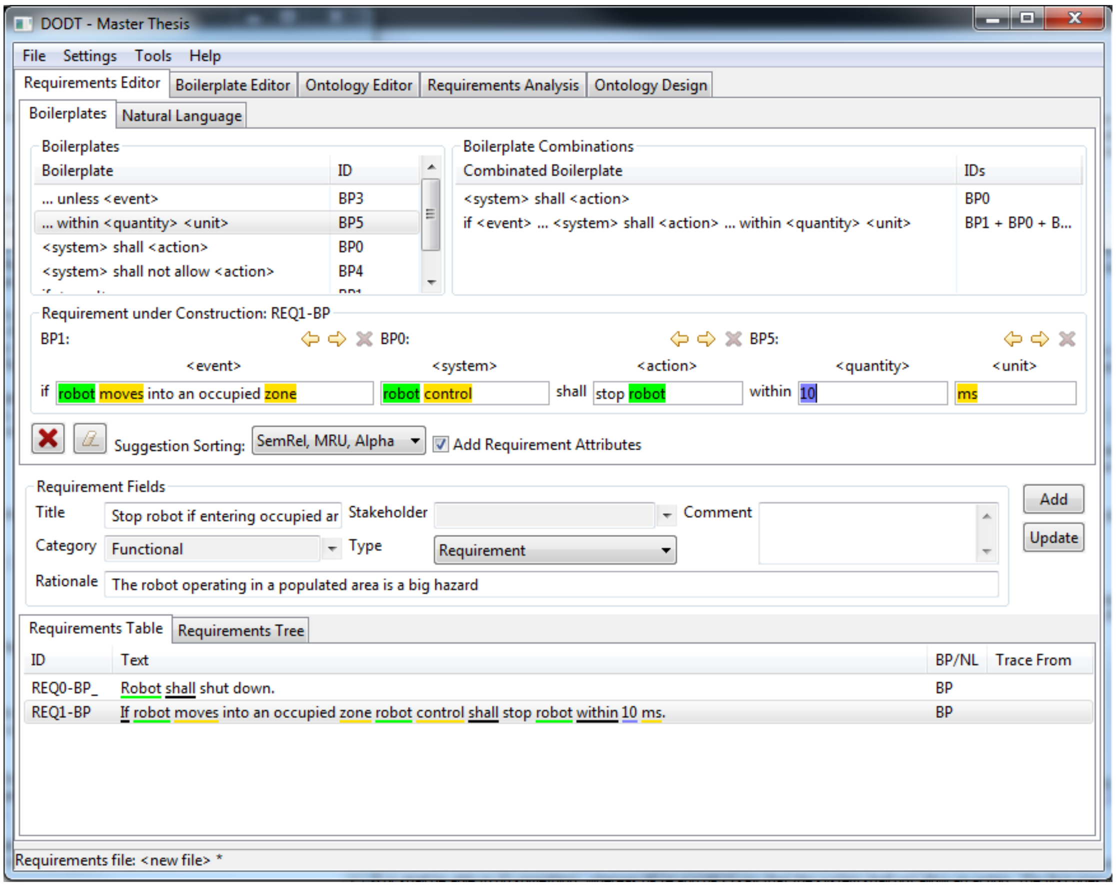
2.2.3 Drawbacks when using boilerplates

Reduced flexibility The boilerplates reduces the freedom to express the requirements as wanted. There is the possibility that constraining the expression may lead the requirements engineers to take shortcuts that may lead to loss of detail and functionality.

Stricter language Boilerplates, though less formal than UML-diagrams, are more formal than free text. Carew et al. [2005] may then suggest that they are more difficult to understand than free text, though several of the benefits listed above may leverage against this effect.

2.2.4 Tools

DODT is a requirements engineering tool built to write and manage boilerplate requirements. The tool allows the creation and customization of ontologies, boilerplate database and a requirements specification. The requirements editor is shown in Figure 2.3. The requirement currently being edited is listed in the middle of the screen, with editable fields for all the attributes.



Boilerplates

Boilerplate	ID
... unless <event>	BP3
... within <quantity> <unit>	BP5
<system> shall <action>	BP0
<system> shall not allow <action>	BP4

Boilerplate Combinations

Combinated Boilerplate	IDs
<system> shall <action>	BP0
if <event> ... <system> shall <action> ... within <quantity> <unit>	BP1 + BP0 + B...

Requirement under Construction: REQ1-BP

BP1: BP0: BP5:

	<event>	<system>		<action>		<quantity>	<unit>
if	robot moves into an occupied zone	robot control	shall	stop robot	within	10	ms

Suggestion Sorting: SemRel, MRU, Alpha
☑ Add Requirement Attributes

Requirement Fields

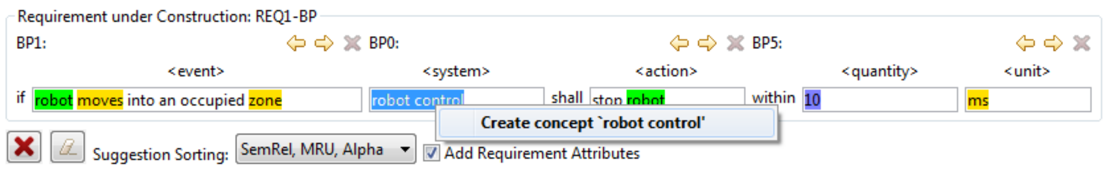
Title	Stop robot if entering occupied ar	Stakeholder		Comment	
Category	Functional	Type	Requirement		
Rationale	The robot operating in a populated area is a big hazard				

ID	Text	BP/NL	Trace From
REQ0-BP_	Robot shall shut down.	BP	
REQ1-BP	If robot moves into an occupied zone robot control shall stop robot within 10 ms.	BP	

Requirements file: <new file> *

Figure 2.3: DODT Requirements editor

The tool monitors whether the attributes are in accordance with the ontology. For example, notice the yellow background for "control" in the system attribute. Neither "Robot control" or "control" is not a part of the ontology, so DODT alerts the requirements engineer, and he or she can quickly create the “Robot control” concept. (See Figure 2.4)


Requirement under Construction: REQ1-BP

BP1: BP0: BP5:

<event>	<system>		<action>		<quantity>	<unit>
if robot moves into an occupied zone	robot control	shall	stop robot	within	10	ms

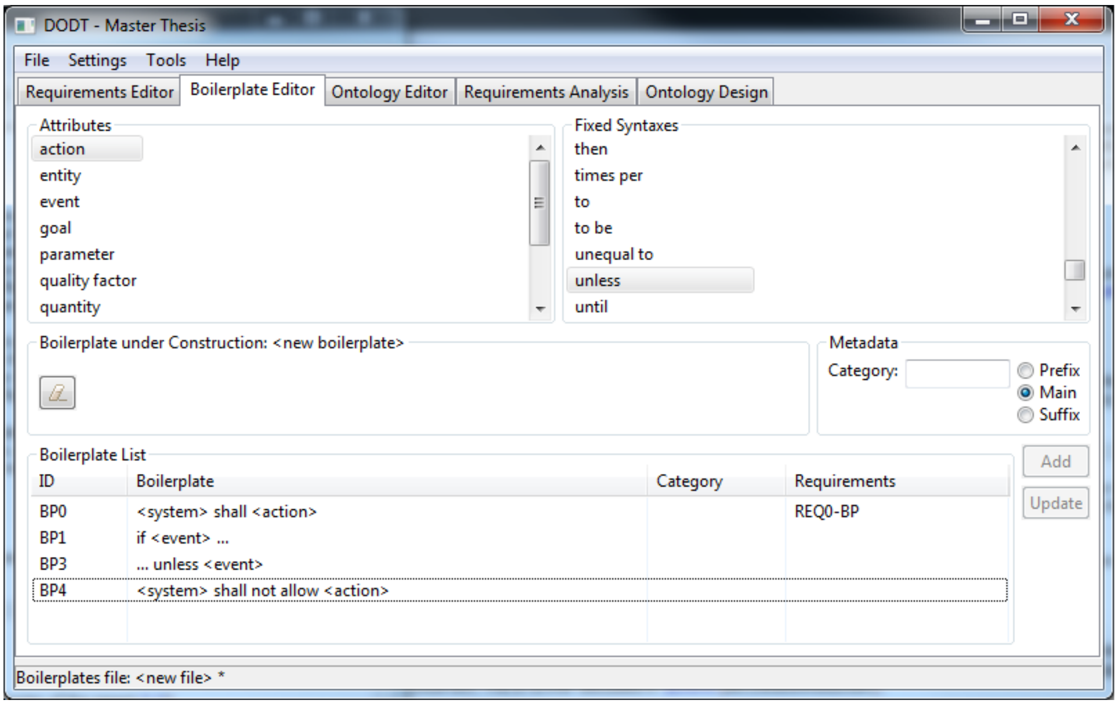
Create concept `robot control'

Suggestion Sorting: SemRel, MRU, Alpha

[✓ Add Requirement Attributes]

Figure 2.4: Creating a concept in DODT

As mentioned at the beginning of the section, DODT enables the user to create and edit boilerplates (Figure 2.5). This enables an organization to create its own set of boilerplates specialized for them.



Attributes: action, entity, event, goal, parameter, quality factor, quantity

Fixed Syntaxes: then, times per, to, to be, unequal to, unless, until

Boilerplate under Construction: <new boilerplate>

Metadata

Category: Prefix [✓ Main] Suffix

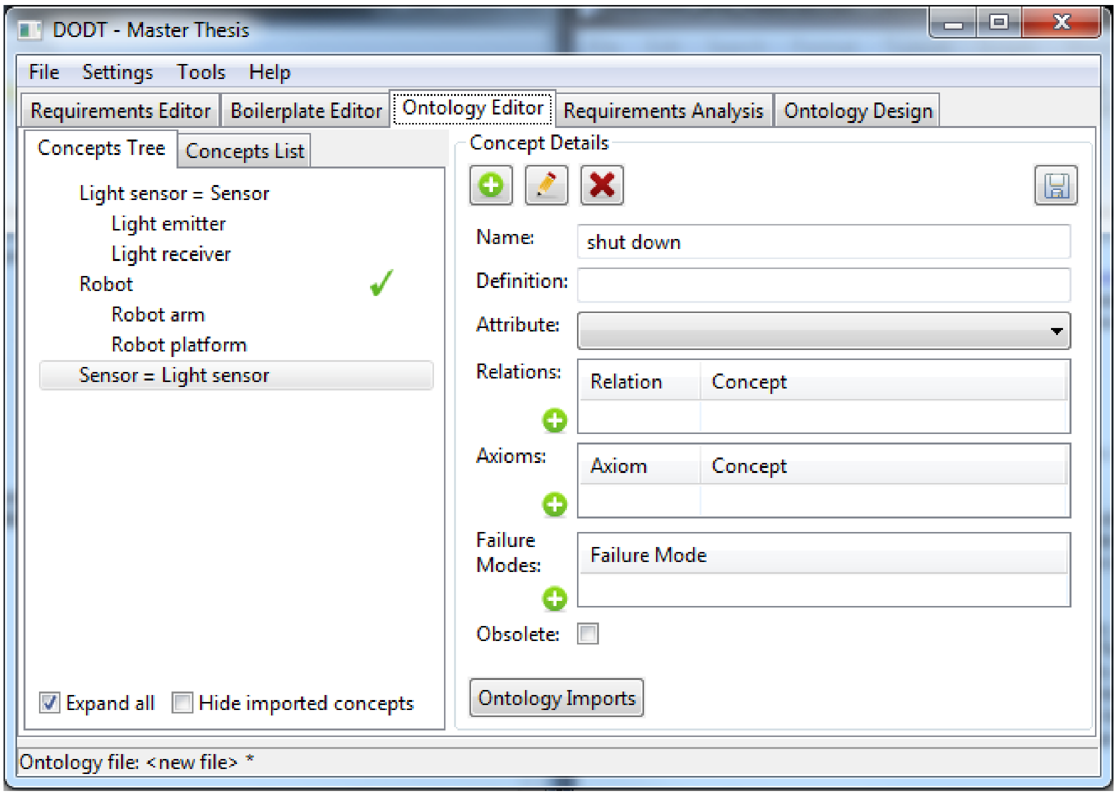
Boilerplate List

ID	Boilerplate	Category	Requirements
BP0	<system> shall <action>		REQ0-BP
BP1	if <event> ...		
BP3	... unless <event>		
BP4	<system> shall not allow <action>		

Boilerplates file: <new file> *

Figure 2.5: Boilerplate editor in DODT

It is also possible to create or import ontologies (Figure 2.6), which are used by the tool to analyze completeness, inconsistency, ambiguity, noise, opacity, similarity and obsoleteness in the requirements.



Light sensor = Sensor
Light emitter
Light receiver
Robot ✓
Robot arm
Robot platform
Sensor = Light sensor

Name:	shut down
Definition:	
Attribute:	
Relations:	Relation / Concept
Axioms:	Axiom / Concept
Failure Modes:	Failure Mode
Obsolete:	☐

Ontology file: <new file> *

Figure 2.6: Ontology editor in DODT

2.3 Free Text Requirements

Free text requirements is the traditional method for writing requirements. It is an informal method, utilizing a natural language. There are basically no rules to writing requirements using free text, which makes it easy to formulate requirements. Below is an example of a free text requirements:

The user shall be able to use a simple panel to define time intervals.

The benefits and drawbacks of using free text to formulate requirements are listed in subsection 2.3.1 and subsection 2.3.2.

2.3.1 Benefits to using free text requirements

Flexible Free text requirements will not put any restrictions on how the requirements are written, and therefore has a lower threshold for a person to learn to write requirements in.

Easy to read Free text has the advantage that it uses a natural language, and as such can be very well understood by any stakeholder.

Easy to understand Since free text is just natural language, no training is neccessary to understand a requirement written using free text. This creates a low threshold for communicating with customers.

2.3.2 Drawbacks to using free text requirements

Does not prevent disambiguities or inconsistencies While not putting any restrictions on how the requirements are written, the method does not do anything to prevent poor requirements engineering either.

Difficult for machines to read Because machines are bad at reading natural language, automatic analysis of the requirements specification is difficult.

Lacks uniformity There is neither any inherently standard way of writing the requirements nor a consistent dictionary to make sure similar requirements are written the same way.

No common understanding of concepts Connecting the two previous points, a lack of uniformity makes the requirements "prone to ambiguous representations and inconsistencies".

Chapter 3

Methods and materials

3.1 Context selection

An experiment can be run either on-line or off-line, meaning it is either carried out in a real project setting or is simulated, respectively. An on-line experiment will produce results with better general validity, but will have greater cost and risks connected to it, making it a less attractive choice for this research. In the case of further research, however, an on-line experiment would be benefitial.

For the same reasons as we chose to run the experiment off-line, the test subjects will be third year computer science students. This is cheaper than using professionals, and also involves less risk. The drawbacks are that choosing students as test group will result in a more homogeneous group, making the experiments less generally applicable. They will also have less experience with requirements engineering. However, as one of the main focuses of the experiment is checking how well the test subjects understand boilerplates compared to free text requirements, this may actually be an advantage. The lack of experience may result in more similar conditions for the two methods, making the results more reliable. Getting a sizable group of students is also easier than getting the same number of professionals.

The experiment will use two cases. Both provide previously defined requirements, which are fully valid by industry standards. To limit the size of the tasks, however, only a small part of the requirements are included. Both cases will be described to the test subjects both orally, in text and with a system sketch.

The first case is a heating oven, with a heat sensor, an I/O panel and a controller. This case will be used for the test subjects to find errors in the requirements. The errors are introduced in the originally valid requirements.

The second case is a robot, which can move on a rail between two areas of a tool cell. The safety system shall make sure the robot never operates while someone is in the same area as the robot, and uses a set of sensors to detect movement in and out of the areas. This case will be used for the test subjects to write requirements using boilerplates. A short, and in some cases vague, description of what the requirement should say is provided, and it is then up to the test subjects to formulate correct and precise requirements.

The experiment uses two different cases, from widely different industries. The heating oven is an everyday object, while the robot covers a more industrial system. Although these are two specific cases, the results from the experiment should be generally applicable for the following reasons. First, the cases cover two different industries, scales and complexities. Second, the experiment is designed in such a way that the subjects will solve the same tasks using both methods. (see section 3.5) Thus, the methods will be tested under the same conditions, with regards to both the case and the experience gained from solving previous tasks. Third, it is the quality of boilerplate requirements relative to free text requirements that is examined. This means that it is not important whether the cases are general or specific, because it is the difference between how well the test subjects performed with each method that is evaluated, as opposed to if the experiment were to give some fixed "grade" to the methods. In the latter example a specific, and thus unknown, case would probably lead to poorer results than a more general and well-known case. In this case, both methods use the same case, so the methods used is the only variable, thereby making the results generally valid.

3.2 Hypotheses

The hypotheses presented in this section is based on the research questions presented in section 1.2. They are organized according to the research question they are based on, in order to visualize the "red line" from the intent, through the design of the experiment, ending with the data analysis in chapter 5.

While analyzing the results, the goal will be to disprove the null hypothesis, then to confirm the alternative hypothesis HX_1.

The null hypotheses will be enumbered HX_0, X being the number of the hypothesis, while the alternative hypotheses will be enumbered HX_1 or HX_2.

Research question 1

Is it easier to prevent ambiguities when writing boilerplate requirements than free text requirements?

$H1_0$ There is the same or a higher number of ambiguities in requirements written using boilerplates as using free text requirements.

$H1_1$ There are fewer ambiguities in requirements written using boilerplates than using free text requirements.

Research question 2

Is it easier to discover ambiguities in boilerplate requirements than in free text requirements?

$H2_0$ The same or a lower percentage of ambiguities are discovered in requirements written using boilerplates as using free text requirements.

$H2_1$ A higher percentage of the ambiguities are discovered in requirements written using boilerplates than using free text requirements.

Research question 3

Is it easier to prevent inconsistencies when writing boilerplate requirements than free text requirements?

$H3_0$ The same or a lower percentage of inconsistencies are discovered in requirements written using boilerplates as using free text requirements.

$H3_1$ A higher percentage of the inconsistencies are discovered in requirements written using boilerplates than using free text requirements.

Research question 4

Is it easier to discover inconsistencies in boilerplate requirements than free text requirements?

$H4_0$ The same or a lower percentage of inconsistencies are discovered in requirements written using boilerplates as using free text requirements.

$H4_1$ A higher percentage of the inconsistencies are discovered in requirements written using boilerplates than using free text requirements.

Research question 5

How much introduction to boilerplates is necessary to both understand and be able to write new requirements using boilerplates?

$H5_0$ Extensive training is required in order to understand and write requirements using boilerplates.

$H5_1$ An introduction of up twenty minutes is sufficient to understand and write requirements using boilerplates.

3.3 Variables

The independent variable of interest in this study is the requirements engineering method: boilerplates and free text. It is the effect of these treatments the experiment will try to measure.

Other independent variables, which will not be studied in-depth in this project, are length of education, work experience and length of training. These are mentioned in subsection 5.1.1, and discussed in chapter 6, but will not be tested in this experiment.

The dependent variables are the number of errors produced when writing requirements and the number of errors found when inspecting requirements.

3.4 Test subjects

The test subjects were recruited from the third year of the Computer Science studies at NTNU, and volunteered in order to collect money for a class trip. The trip commitee receives 250 NOK in compensation for each student participating. A total of 39 students participated, being evenly distributed on the two treatments described in section 3.4.

Most of the participants have two years of studying as their only experience in software engineering. With respect to requirements engineering, they were currently taking the cource TDT4140 Software Engineering, but were only halfway through the course at the time of the experiment. Because of this, their experience were limited, and the expectations for their performance must be adjusted according to this. The upside is that they are not significantly more experienced using either method. A professional developer would be trained in writing and understanding free text requirements, and this would affect the expected results in favour of free text requirements.

3.5 Experiment Design

In the first, practical part (section A.2) of the experiment, the participants are asked to find errors in requirements. One half of the requirements are written using boilerplates and the other half is written using free text. In this context, a number of threats to the validity of the experiment arise: The subjects could learn from evaluating the requirements written using either method, so that they have a deeper understanding of the system when they evaluate the requirements written using the second method. Or they could become more experienced with reading requirements, especially considering their previous experience is limited. These threats are further discussed in **??**.

To reduce or eliminate the threats, the order of the tasks was alternated. Half of the test subjects did the tasks concerning boilerplate requirements first, while the other half did the free text requirements first. Any possible learning would then be evenly distributed on the two methods. Which participants used which treatment first, was determined by which side they sat down on. The distribution between the treatments was as even as possible; 20 for one and 19 for the other.

3.6 Instrumentation

Before the experiment started, the test subjects were given a 20 minutes oral presentation about boilerplates and requirements engineering in general.

All the tasks in the experiment was collected in a booklet. This booklet also contained descriptions of the two systems used in the cases, as well as a list of all the boilerplates.

Chapter 4

Operation

4.1 Preparation

The experiment was carried out in an auditorium, with the participants sitting on alternate rows and with a minimum of two seats between each participants, thus preventing them from peeking at each other's answers. The use of the auditorium also made it possible to use a projector to show a PowerPoint presentation (??) as part of the introduction.

4.2 Execution

At the start of the experiment, the test subjects were given a 20 minutes oral introduction to software requirements in general, the differences between boilerplates and free text requirements and also the two systems used in the experiment. The PowerPoint presentation used during this introduction is included in ??.

The first part presented what software requirements are, what quality measures there are in relation to software requirements and how poor requirements influence the results of a project. Some example requirements were given as well.

The second part presented the two methods this project is concerned with, and explained the differences between boilerplates and free text when writing requirements.

Finally, the heating oven and the robot arm systems were presented briefly. During this presentation the subjects were allowed to ask questions about anything that was confusing or unclear, both about the methods and about the systems presented, while questions posed while they were doing the tasks in the experiment were only answered if they related to errors in the tasks, such as typos and changes from earlier versions of the experiment that had not been altered. Few questions were asked during the presentation. The test group claimed to understand both the methods and the case pretty well.

The test subjects were given up to one hour to do all the tasks in the experiment. This included answering the pre- and post-experiment questionaires. They were allowed to turn in the answers and leave when they had finished, as long as all tasks had been done properly. Only one out of the 39 delivered before 45 minutes had passed, but since that person had done all the tasks, this does not seem to have been a problem.

The tasks were given in a booklet (see Appendix A), and consisted of five parts. The first part was a short questionaire about the participant's previous experience, used to define their background. In the second part (see **??**), the participants were asked to find errors in requirements. They should mark the error as either an inconsistency or ambiguity, and also to explain what or why it was a mistake. Thus, it was possible to go back and check that they had actually found an error. This part was split into two parts; one part with boilerplate requirement, and one part with free text requirement.The third part was about writing requirements using boilerplates. The participants were given a case description and also a short description for each requirement. They were then asked to use this information and the boilerplates listed in section A.5 to write three requirements. The last part (see section A.4) was a questionaire about how the participant thought he or she had done on the tasks, more specifically how easy or difficult it was to use the methods, and whether they would prefer one method over the other.

4.3 Data Validation

All of the 39 participants completed the experiment, answering all the tasks. The participants also used most of the time they had available, suggesting that they did their best and did not deliver early, even though they were allowed to leave when they had delivered.

A couple of questions needed to be clarified during the experiment. First, in the explanation for the part where they were to locate errors in requirements, if said that there where four error types, though only two were described. The correct number were two, and as this were explained in plenary, it was assumed to not be an important mistake.

The second issue was question 6 in the post-experiment questionaire, asking whether it was more difficult to write requirements using boilerplates than using free text. This was a leftover from an earlier version of the experiment, where the participants were also asked to write free text requirements. The students were asked to change the phrasing to "It was difficult to write requirements using boilerplates". Four participants did not answer the question, probably because of the confusion surrounding what the question asked. After due consideration, it was decided to keep the question anyhow, as the ones who answered it did answer according to the new question. This was also one of the more important questions to answer.

Chapter 5

Research Results

5.1 Descriptive Statistics

The data from the experiment (included as Appendix A) was of many forms. Section A.1 and section A.4 both had easily quantifiable results, since they both consisted mainly of statements that the test subjects should take a position on by marking one of five boxes, representing positions from "Strongly agree" to "Strongly disagree".

However, the quality of requirements are not as easily tested. Section A.2 tested how well the subjects understood requirements by asking them to find errors, and all answers to the questions in this section had to be verified by an explanation of the errors in the requirements. Thus, these explanations had to be evaluated, before the proposed error was accepted. Similarly, in section A.3, all written requirements had to be evaluated in order to judge their correctness.

5.1.1 Previous Experience

As explained in section 4.2, the participants were asked to answer a few questions about themselves before starting on the experiment tasks. In two of the questions they were asked to state to which degree they agreed or disagreed with statements about their previous experience with requirements specification in general and boilerplates in particular.

As shown in Table 5.1, a total of 64% agree either partially or strongly with the statement that they have no previous experience with requirements specification, whereas 23% disagree partially. This can be seen in correlation with the answers to whether the participants had taken the course TDT4242 Requirements

Statement	Strongly agree	Partially agree	Neutral	Partially disagree	Strongly disagree
”I have no previous experience with requirements specification”	31%	33%	13%	23%	0%
”I have no previous experience with using boilerplates”	85%	13%	3%	0%	0%

Table 5.1: How the participants evaluated their previous experience

and Testing (which teaches the boilerplates method) or any related courses. None of the participants had taken the aforementioned course, but 28% answered that they had taken the course TDT4140 System Development, which teaches requirements specification. However, since the participants are all from the same class, it is likely that the remaining 72% have also taken this course. The course was being taught at the time of the experiment, which may explain why the students answered that they did not have experience with requirements specification.

5.1.2 Understanding Requirements

In Task 1 and 2 the test subjects were asked to find errors in 11 requirements. Six of the requirements were written using boilerplates, while five were written in free text. They were to mark each incorrect requirement as either inconsistent or ambiguous, and briefly describe the error. This last part was included to ensure that what they found was actually an error, and to force the test subjects to really think about what they checked off. The answers were then evaluated by the author of this paper and his supervisor, Professor Tor Stålhane, first individually, and then discussing their evaluation.

Although the errors were described in several different ways, it was possible to categorize them. This was done by first reading through all the answers, noting the different answers. Then they were grouped together, and one common description of the error was made. These are listed in tab:ValidErrorsInBPRequirements. After determining all of the valid errors, the answers were inspected again, marking them as either correct or incorrect. The thought behind using this process was that by determining which answers would be accepted before starting to evaluate them, it was ensured that the way they were evaluated would not change during the process.

Requirement	Error-type	Error
BPR1	-	-
BPR2	Ambiguous	It is not specified which type of sensor is meant
BPR3	Inconsistent	The heating unit, not the system, shall have sensor.
	Inconsistent	It is not specified what should be sampled from the sensor.
BPR4	Inconsistent	This does not differentiate between whether the temperature is too high or too low. If the temperature is > d higher than T, system shall switch off heating unit.
BPR5	-	-
BPR6	Inconsistent	T0 is the default temperature, and should be applied only for the time outside of the specified time intervals.
	Inconsistent	In conflict with the requirement that user shall be able to set the temperature for the defined intervals.
	Ambiguous	Incomprehensible

Table 5.2: Valid errors in boilerplate requirements

For the incorrect requirements written using boilerplates, on average 47% of the participants found the errors. Although the participants were asked to separate between inconsistent and ambiguous requirements, this report will not separate between the two, because some requirements, like BPR6, are marked as both. Some participants described a concrete error in the requirement (inconsistency), while others only marked it as unclear (ambiguous). In my opinon, there is a high probability that they refer to the same one, because there are no other suggested errors for this requirement. Because of this, it does not make sense to separate between the two.

	BPR1	BPR2	BPR3	BPR4	BPR5	BPR6
# of inconsistencies	-	-	51%	62%	-	54%
# of ambiguities	-	10%	-	-	-	13%

Table 5.3: The percentage of the participants who found errors in requirements written using boilerplates

For BPR6, it varies whether they are marked as ambiguous or inconsistent. Most of the participants (46%) called it inconsistent that the temperature during user-defined time intervals should be set to the default temperature, thus conflicting with the requirement that users should be able to set the temperature. However, 21% of the participants marked the requirement as incomprehensible, thereby calling it ambiguous.

	FTR1	FTR2	FTR3	FTR4	FTR5
# of inconsistencies	-	-	-	69%	-
# of ambiguities	5%	-	-	3%	-

Table 5.4: The percentage of the participants who found errors in free text requirements

For the requirements written using free text, 38% of the participants found the errors. However, two factors influence this number. Only two of the five requirements were deemed incorrect, making the basis of the evaluation limited. For the requirements written using boilerplates, four out of six were incorrect, giving a more solid sample size. The other factor, which is more interesting, is that FTR1 was only marked as incorrect by two of the participants. The rest either had nothing to remark, or suggested as faults things that were not actually incorrect. Because one of the two requirements was only marked incorrect by 5%, greatly influences the results. 72% marked FTR4 incorrect. This is further discussed in section 5.2.

Requirement	Error-type	Error
FTR1	Ambiguous	Shall the temperature be set *for* the interval or *during* the interval?
FTR2	-	-
FTR3	-	-
FTR4	Inconsistent	If the temperature is too high, the heating element should be shut off, not on.
	Ambiguous	Incomprehensible
FTR5	-	-

Table 5.5: Valid errors in free text requirements

5.1.3 Writing Requirements

In part three of the experiment (section A.3) the participants were asked to write requirements using boilerplates. The answers were evaluated according to five categories; correct use of boilerplates, preserved meaning, simplicity, inconsistency and ambiguity. The criteria for each category to be to correct is listed in Table 5.6. A summary of the results are shown in Table 5.7.

Evaluation criteria	Explanation
Correct use of BPs	Appropriate boilerplates are used. Prefix- and postfix boilerplates are used where neccessary, and the boilerplates does not reduce the scope of the requirement. Closely connected to "Preserved meaning".
Preserved meaning	The scope and meaning of the requirement is not reduced or changed. There are no elements in the requirement that is not described in the task description.
Simplicity	Splits up requirements where possible, to make each one as independent and simple as possible. This makes the requirements easier to test.
Consistent	The requirement does not contradict either itself, the system description or the task description. Use appropriate names for entities, etc. Functionality is delegated to appropriate subsystems.
Unambiguous	The requirement is clear and easily understandable. The requirement is not open for several interpretations.

Table 5.6: Evaluation criteria of written requirements

	Correct use of BPs	Preserved meaning	Atomic	Inconsistent	Ambiguous
% Correct Reqs	66%	26%	95%	62%	69%
% Incorrect Reqs	34%	74%	5%	38%	31%

Table 5.7: Results from evaluation of requirements written using boilerplates

As we see in Table 5.7, for 34% of the requirements written, the participants were not able to use appropriate boilerplates. Most of these mistakes were made in Task 5. 82% of the participants used correct boilerplates for requirements 1 and

2, whereas only 33% for requirement 3. One of the most frequent mistakes were mixing up BP16 or BP19 for BP24 or BP25. BP16 and BP19 say that the system shall do or shall be able to do something, whereas BP24 and BP25 say that the system shall not allow an action. The first ones were used in combination with one of the if-prefixes, while the latter ones could be used with either an if-statement or (preferably) the unless-postfix. The difference is subtle, but definitive. Saying what the system shall not allow states that there shall be a control mechanism, while only saying what it shall do in certain cases implies no separate control.

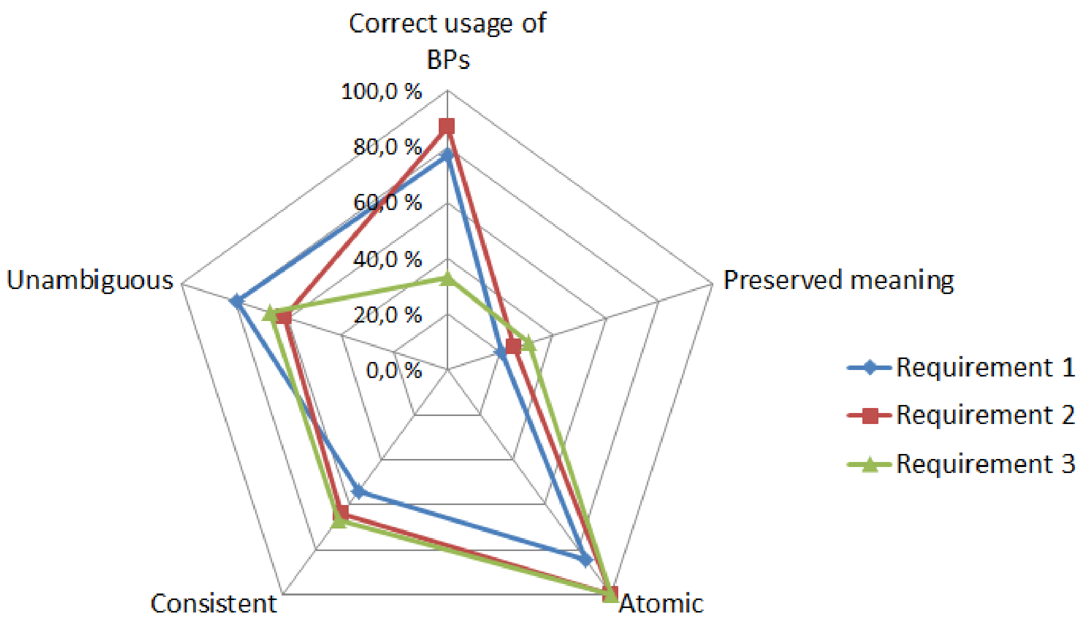


Figure 5.1: Correctness of the written requirements according to each evaluation criteria

However, the results are more interesting if we compare results for each of the three requirements the participants were asked to write. As Figure 5.1 shows, requirement 3 stands out significantly when it comes to how many used the correct boilerplates. Whereas the percentage of correct usage for Requirement 1 and 2 were 77% and 87%, respectively, the corresponding result for Requirement 3 was only 33%. The standard deviation for these results are as high as 23.3%. For the other evaluation categories, the results are more uniform, and the standard deviations for all of the other categories are below 7.5%. Subsection 6.2 discusses why the results in one category differ so much from the other requirements.

5.1.4 Self-evaluation

The final part of the experiment was an evaluation form. The subjects were asked to take a stand on ten statements concerning how they evaluated their own performance and what their opinion was about the two methods after the experiment. They had to mark whether they strongly or partially agreed or disagreed with the statement, or whether they neither agreed nor disagreed (named "Neutral" on the form). The results are listed in Table 5.8 and discussed in connection with the other results in chapter 6.

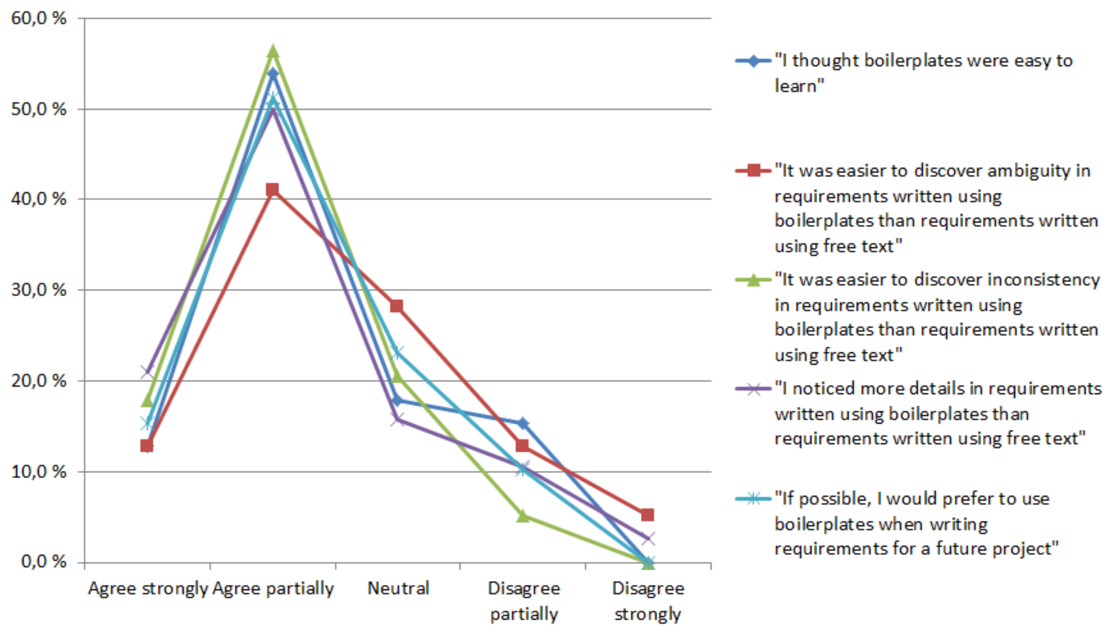


Figure 5.2: Results from positive self-evaluation statements

The participants were overwhelmingly positive to the boilerplate method. As the graphs in Figure 5.2 shows, more than half of the participants agreed partially or strongly to all the positive statements about boilerplates. We even see that the graphs for most of the statements follow the same curve, except for the one that said "It was easier to discover ambiguity in requirements written using boilerplates than requirements written using free text". Still, this one also were agreed to by the majority of the respondents.

At the other end of the scale, we take a look at the statements with a negative look on boilerplates. To some degree, these follow the opposite of the graphs of the

	Agree strongly	Agree partially	Neutral	Disagree partially	Disagree strongly
”I thought boilerplates were easy to learn”	5	21	7	6	0
”I thought boilerplates was difficult to learn”	0	7	8	17	7
”It was easier to discover ambiguity in requirements written using boilerplates than in requirements written using free text”	5	16	11	5	2
”It was easier to discover inconsistency in requirements written using boilerplates than in requirements written using free text”	7	22	8	2	0
”Requirements written using boilerplates was more difficult to understand than requirements written using free text”	1	11	5	14	7
”It was difficult to write requirements using boilerplates”*	1	11	9	12	2
”I noticed more details in requirements written using boilerplates than in requirements using free text”	8	19	6	4	1
”I did not see a difference between requirements written using boilerplates and requirements written using free text”	0	0	0	14	24
”If possible, I would prefer to use boilerplates when writing requirements for a future project”	6	20	9	4	0
”If possible, I would avoid to use boilerplates when writing requirements for a future project”	0	3	6	20	10

Table 5.8: Results from selv-evaluation form. (* Changed from ”It was more difficult to write requirements using boilerplates than to write requirements using free text” during experiment. See section 4.3)

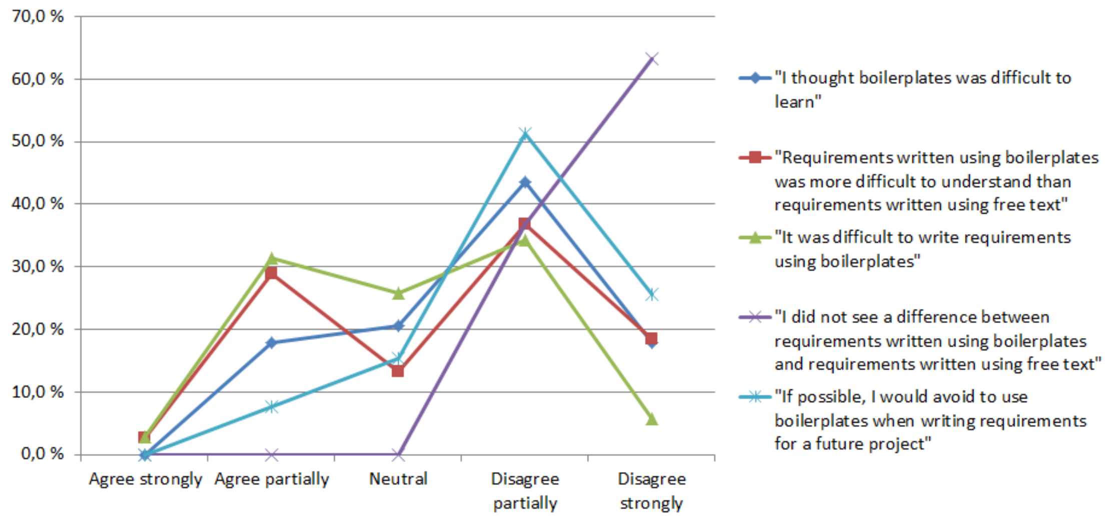


Figure 5.3: Results from negative self-evaluation statements

positive statements. However, there are some exceptions. 29% of the participants partially agreed that ”requirements written using boilerplates was more difficult to understand than requirements written using free text”. And 31% agreed that ”it was difficult to write requirements using boilerplates”. The statement ”I did not see a difference between requirements written using boilerplates and requirements written using free text” is not neccessarily negatively charged, but is still included in Figure 5.3. No one agreed to any degree with the statement, and 63% disagreed strongly.

5.2 Data Set Reduction

As explained in subsection 5.1.2, a number of marked errors in the experiment results were eliminated, because they were incorrect. They were eliminated based on the description of the error.

The findings showed that about 28% of the registered errors were not actual errors. Although this number is high, it was not unexpected. When asked to find errors, the participants will try to find errors even where there are none. By eliminating these incorrectly marked errors, we found the number of participants who had correctly crossed for incorrect requirements, listed in Table 5.3.

From the first two tasks, two requirements stand out. Table 5.3 show that the ambiguity in BPR2 was found by only four of the test subjects, or 10%, whereas Table 5.4 shows that the ambiguity in FTR1 was only found by two test subjects, or 5%. These two errors affect the results to such a high degree, that their inclusion must be discussed.

In BPR2, four of the test subjects pointed out that it was not specified what kind of sensor the system should have. Although it says "In order to measure temperature...", several different types of sensors could be implied, and they are therefore correct in pointing this out. However, since only 10% of the participants found this error, this requirement influences the overall averages for how many of the test subjects correctly marked each incorrect requirement.

For the ambiguity in FTR1, the two subjects who pointed it out, claimed that the part "...set the temperature during this interval" could be interpreted as both "the user shall be able to set the temperature for the defined time interval" and "the user shall be able to set the temperature for the interval during the interval". Granted, the latter would not be a very sensible implementation, but because the wording can be interpreted in two ways, the requirement is ambiguous. That only two participants pointed this out, may mean that the rest found the second interpretation too unlikely, or that they simply overlooked this detail, but it also shows that two out of 39 found this ambiguous.

With the inclusion of both requirements, and the number of test subjects finding the respective ambiguities, the numbers show that on average 47.4% of the subjects correctly marked incorrect boilerplate requirements, while 38.5% correctly marked incorrect free text requirements, giving an edge to boilerplates. However, if we remove the two requirements, with the argument that the errors were too small, the numbers shift completely. Though the average number of test subjects who correctly marked incorrect boilerplate requirements is raised to 59,8%, the corresponding number for free text requirements is raised to 71,8%, flipping the edge to free text.

Chapter 6

Evaluation

6.1 Understanding Requirements

Discovered Differences Between BPRs and FTRs

The results presented in subsection 5.1.2 suggests that it is easier to discover errors in boilerplate requirements than in free text requirements, as the average participant in the experiment found 47% of the errors in the boilerplate requirements, but only 38% of the errors in the free text requirements. However, several factors have to be taken into account when analyzing the results.

First, some requirements greatly influenced the results. Only 10% of the participants marked BPR2 as incorrect, and that was actually better than the 5% that marked FTR1 as incorrect. We will get to the reason why so few found these errors below, but first we will see how the results would look without these two requirements. When removing the two, the average number of errors found in boilerplate requirements rises to 60%, while the same number for free text requirements jumps to 72%. The big leaps are the results of a relatively small sample size. Other than BPR2, there are three incorrect boilerplate requirements, and there is only one incorrect free text requirement in addition to FTR1. Before the experiment was carried out, more of the requirements were thought to be incorrect, but this view was changed after evaluating the results. Some of the "errors" that had been inserted into the requirements were too small to consider the requirements incorrect. The limited sample size raises the uncertainty about the validity of the conclusions. Other factors, like the type of error and the complexity of the requirement, may have had a bigger impact on how many found the error than whether the requirement was written using boilerplates or free text.

So, which method is better? If the results lean one way when using all the data, and the complete other way when removing the outliers, can we say anything for certain?

One thing is clear: there are other factors influencing the results in addition to which method is used to write the requirements. We have already mentioned the type of error and complexity as other factors in this experiment. Experience looks to be just as important. The participants were all from the second year of the Computer Science program, and had barely started learning about requirements engineering. The low percentage of errors found, even using the higher estimates when removing the two requirements that stand out, suggests that they were too inexperienced. Although this means that neither method should be expected to be favoured, it also means that the overall results should not be expected to be very high.

Complexity

One factor which may have been influential as to how well the requirements were understood, and thus how many of the errors were found in them, was the complexity of the requirement. Take the free text requirements, for instance. 72% noticed that FTR4 was incorrect, while only 5% found the error in FTR1. FTR4 is a relatively simple requirement, consisting of an if-condition and the action which shall be taken if the condition is true. FTR1, however, is a compound requirement, making the error more difficult to spot. Whether the same is true for boilerplate requirements is difficult to tell, because the boilerplate requirements used in this experiment were not complex enough to compare. Theoretically, since boilerplate requirements are so close to natural language when instantiated, the observations from free text requirements should be transferable. If so, errors in boilerplate requirements on the whole should be easier to discover, because the nature of boilerplates prevents the requirements to become complex.

Type of Errors

In addition to the complexity of the requirement, the different types of errors also appear to have an effect on how many discover it. BPR4, BPR6 and FTR4 are all incorrect because of functional inconsistencies. BPR4 does not differentiate between whether the temperature is too high or too low. BPR6 says that the temperature shall be set to the default temperature during all [user-defined] time intervals. FTR4 has the same error as BPR4. It was in these three that the highest percentage of participants (an average of 67%) found errors.

There are three other incorrect requirements. Two have already been discussed in above; BPR2 and FTR1. The error in BPR2 is related to which type of sensor is meant, while in FTR1 it is unclear whether the user shall be able to set the temperature for an interval, or have to set it during the interval. Both of these are ambiguities, and were marked by only an average of 8%. The final incorrect requirement is BPR3, which has two errors; "heating unit" should have been "system", and it was not specified what the sensor shall sample. The percentage of participants who marked this requirement was higher than for BPR2 and FTR1, but still lower than the three mentioned above. As we see, none of these three incorrect requirements are functionally inconsistent. BPR3 inconsistently use the heating unit in stead of the system, while the

6.2 Writing Requirements

In section 6.2, it was shown that the overall requirements written by the participants contained many mistakes. The most glaring one was the test subjects' failure to preserve the meaning from the task description when writing the requirements. Only 26% of the written requirements were completely in line with what the task description specified.

Part of the reason for the poor results should be attributed to the participants' limited experience. Most of them were only in their second year of the Computer Science studies, and less than a fourth of them had relevant work experience. As such, the expectations should be lowered.

However, if looking at each requirement individually, we can see certain differences that need to be explained. For example, why did so few of the participants manage to choose a suitable boilerplate for Requirement 3 (section A.3)? Three types of errors stand out from the answers.

One is the subtle, yet important, difference between saying "the system shall not allow" some action and "the system shall be able to" or "the system shall not" do some action if or while some condition is fulfilled. The first formulation implies some sort of control mechanism. The other formulations lack this control mechanism.

The second error is the two latter examples' use of if- or while-postfixes, which resulted in many situations were the requirement did not specify what to do. As an example, "If <gate A = gate B = closed, and reset switch = on>, <robot control> shall allow <robot> to be <on>" only specifies a condition where the robot shall be able to run. It does not say that the robot shall not be allowed to run if the if-conditions are not fulfilled.

The third type of error is the use of “In order to”. This is a prefix intended to either specify the goal of a requirement (BP30) or to state that the requirement is a prerequisite for another requirement (BP29). Most of the participants who used “in order to”, used BP29. However, it seems as if they did not consider the implications of the requirement. The participant who wrote “In order to <start>, <system shall have both doors closed and register reset switch pressed>” probably had the right idea, but this can be interpreted like the doors shall always be closed, so that it can start, not just that they have to be closed when starting.

Another mistake that a lot of the participants made, was using the broad term “the system” too much. Particularly in Task 3 and 4, they wrote that if the right preconditions were fulfilled, “(...) <system> shall <shut down> (...)”. It would be impractical to shut down the whole system each time the someone enters the area the robot is operating in. Also, since the task description specifies that the robot shall shut down, this mistake is strange. Although no further research was done to find the reason for this generalization, a possible cause may be that the boilerplate uses <system> shall <action> and the participants did not bother to replace system with a more specific term. If this assumption is correct, the use of a requirements engineering tool, such as DODT, should help reduce the frequency of this error.

Chapter 7

Conclusion and Further Work

This project has tested how well test subjects are able to discover errors in requirements written using boilerplates and using free text, and how well they are able to write requirements using boilerplates after receiving a twenty minutes introduction to the requirements engineering method.

The goal of the project was to determine whether boilerplates was a better method for writing requirements than free text. In my opinion, the results do not give a clear answer to this question. Mainly, this is because a couple of the requirements used were too influential on the results. When included, the results showed that boilerplates had a clear edge. However, the errors in these two requirements were found by so few, that it was natural to look at how the numbers changed when removing them from the calculations. Then the results changed completely, and free text looked like the better method.

However, when looking deeper, it seems that the complexity of the requirements had a lot more to say about how easy it was to discover errors than which method were used to write them. This does suggest that boilerplates should be prefered, as that method prevents the requirements engineer from writing too complex requirements.

It is also my opinion that using tools like DODT to write requirements using boilerplates will greatly improve the quality of the requirements. (See section 6.2)

7.1 Further work

Based on the results of this project, there are a few areas where further studies are required. First, it is neccessary to see how more experienced test subjects perform.

The ones used in this experiment had no previous experience, and that affected the general results.

Also, it would be important to test how the introduction of ontologies and the DODT tool would affect the quality of requirements.

Bibliography

Carew, D., Exton, C., and Buckley, J. (2005). An empirical investigation of the comprehensibility of requirements specifications. *International Symposium on Empirical Software Engineering (ISESE).*

Hull, E., Jackson, K., and Dick, J. (2009). Requirements engineering.

The Standish Group, t. (1995). Chaos Report. ...

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslen, A. (1999). *Experimentation in software engineering: an introduction.* Kluwer Academic Publishers.

Zojer, H., Farfeleder, S., Passes, A., Tojal, J., Adedjouma, M., Kondeva, A., Keuler, T., de Assis, P. O. A., Rugina, A.-E., Stålhane, T., Daramola, O., Oertel, M., Rehkop, P., Montes, C. A., Rementeria, J. M., Haugen, O., and Loughran, N. (2011). Revised Definitions of Improved RE Methods D SP2 R3.3 M3 Vol 2. ...

Appendices

Appendix A

Experiment

CESAR - text vs. boilerplates

What is more efficient - requirements using free text or using boilerplates (templates)?

TDT4900
Master Thesis, 24.02.2012

Master-student: Vegard Johannessen

Veileder: Professor Tor Stålhane

Fakultet for informasjonsteknologi, matematikk og elektroteknikk
Institutt for datateknikk og informasjonsvitenskap

Innledning

Målet med dette eksperimentet er å sammenligne to teknikker for å skrive systemkrav, for å finne ut om den nye metoden (boilerplates) er bedre enn den etablerte (fri tekst/naturlig språk). De to teknikkene vil sammenlignes ved at testdeltakerene gjennomfører de følgende aktivitetene:

1. Fyll ut informasjon om ditt erfaringsnivå - ca 5 minutter
2. Følg med på presentasjonen om boilerplates og systemet som skal brukes i oppgavene - ca 20 minutter
3. Løs oppgavene - ca 60 minutter
4. Fyll ut evalueringsskjemaet - ca 5 minutter

A.1 Tidligere erfaring

	Veldig enig	Delvis enig	Nøytral	Delvis uenig	Veldig uenig
Jeg har ingen tidligere erfaring med kravspesifiser-ing					
Jeg har ingen tidligere erfaring med bruk av boilerplates					

Har du hatt faget TDT4242 Kravspesifikasjon og test-ing, eller evt. tilsvarende fag?	....
Hvis du har hatt tilsvarende fag, hvilke?	....
Hvor mange år har du studert (universitet/høyskole)?	....
Hvor mange år/måneder med IT-relatert arbeidserfaring har du (sommerjobber, fast jobb, etc)?	....

Oppgaver

Oppgavene som skal utføres er delt i to; først skal du lese og finne feil i en rekke krav, og etterpå skal du få skrive noen krav selv.

A.2 Å lese krav

I den første delen omhandler kravene varmeovn-eksempelet som ble presentert i innledningen. En kort oppsummering av systemet er beskrevet på neste side.

Halvparten av kravene er skrevet i fri tekst og halvparten er skrevet med boilerplates. En del av kravene er inneholder feil, og din oppgave er å finne disse feilene. Kryss av for hvilken type feil du finner i kravet, og gi en kort forklaring. For eksempel:

Krav 1	Inkonsekvent	*X*	Tvetydig	
Forklaring:	*Kravet kaller <systemet> for <ovnen>, som er i konflikt med ontologien.*			

Hopp videre til neste krav hvis du ikke finner noen feil i kravet.
Feiltypene kan beskrives som følger:

Inkonsekvent = Kravet følger ikke navngivingsreglene fra ontologien, kravet er formulert på en annen måte enn tilsvarende krav, kravet motsier seg selv, dvs det er en kontradiksjon (kravet kan aldri oppfylles) eller kravet motsier et annet krav eller systembeskrivelsen.

Tvetydig = Kravet kan tolkes på flere måter.

Dersom kravet ikke er forståelig, men du ikke synes noen av de fire feiltypene passer, så skriver du bare "uforståelig" i forklaringsfeltet. Du får muligheten til å vurdere forståeligheten til kravene i evalueringsskjemaet på slutten.

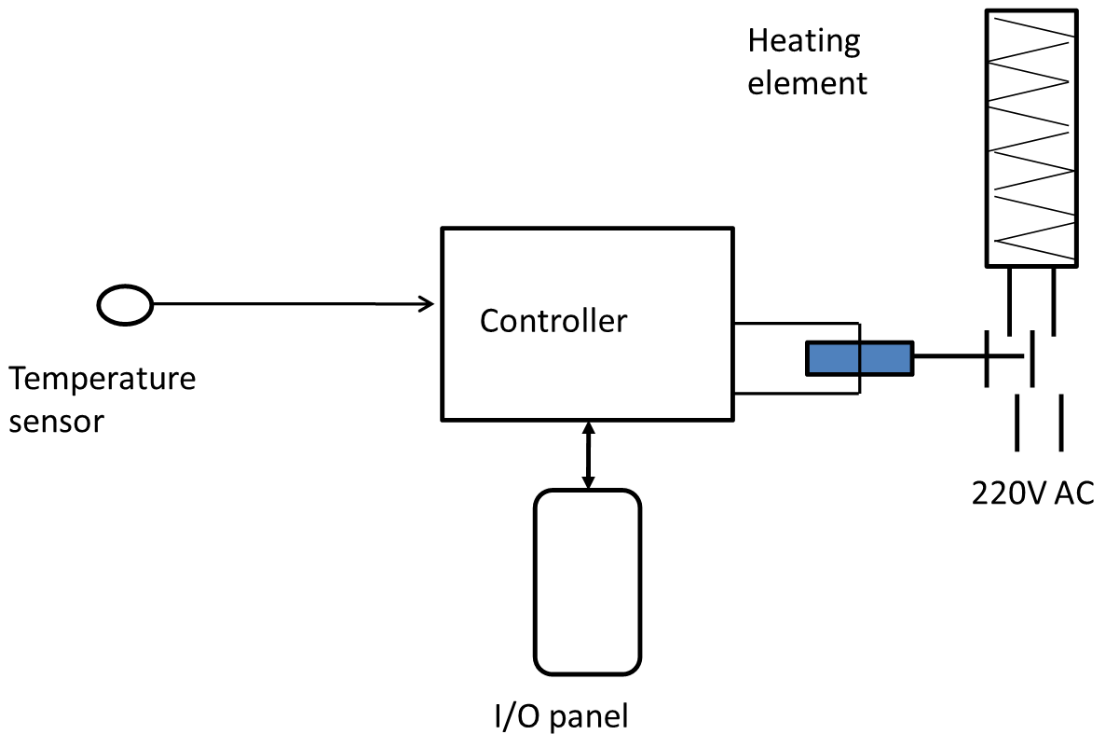
Systemet består av fire enheter;

- Et I/O panel hvor brukeren kan konfigurere systemet
- Et varmeelement (Heating element) som skaper varme
- En sensor (Temperature sensor) som måler temperaturen i luften rundt ovnen
- En kontroller (Controller) som kobler sammen de tre andre enhetene.

Kontrolleren mottar input fra sensoren, og slår på eller av varmeelementet basert på om temperaturen er for høy eller lav i forhold til temperaturen brukeren har tastet inn på I/O panelet.

Brukeren kan stille inn tidsstyring (time intervals) for å sette en fast temperatur til visse tider på døgnet, f.eks. for å senke temperaturen om natten.

Brukeren kan også sette inn en feiltoleranse for temperaturen. For eksempel kan brukeren stille inn at det er greit at temperaturen ligger opp til 2°C over eller under innstilt temperatur. Da vil ikke varmeelementet slås av eller på før temperaturforskjellen overstiger denne verdien.



Boilerplate krav

1. In order to <control temperature>, <heating unit> shall have <on / off switch>
2. In order to <measure temperature>, <system> shall have <sensor>
3. <heating unit> shall <sample> <sensor> at least <3> times per <minute>
4. If <temperature sensor reading differs from T more than d>, <system> shall <switch on heating unit>
5. <system> shall have <default temperature = T0>
6. <system> shall <set> <T = T0> during <all time intervals>.

Oppgave 1: Fyll inn de feilene du finner i kravene over

Krav 1	Inkonsekvent		Tvetydig	
Forklaring:				

Krav 2	Inkonsekvent		Tvetydig	
Forklaring:				

Krav 3	Inkonsekvent		Tvetydig	
Forklaring:				

Krav 4	Inkonsekvent		Tvetydig	
Forklaring:				

Krav 5	Inkonsekvent		Tvetydig	
Forklaring:				

Krav 6	Inkonsekvent		Tvetydig	
Forklaring:				

Fritekst krav

7. The user shall be able to use a simple panel to define time intervals, and set the temperature during this interval - Ti
8. The time intervals are not allowed to overlap
9. The user shall be able to use a simple panel to define a temperature tolerance value d.
10. If the temperature deviates by d, the heater shall be turned on
11. If the user don't set an allowed temperature deviance, the system shall use default value d = 2°C

Oppgave 2: Fyll inn de feilene du finner i kravene over

Krav 7	Inkonsekvent		Tvetydig	
Forklaring:				

Krav 8	Inkonsekvent		Tvetydig	
Forklaring:				

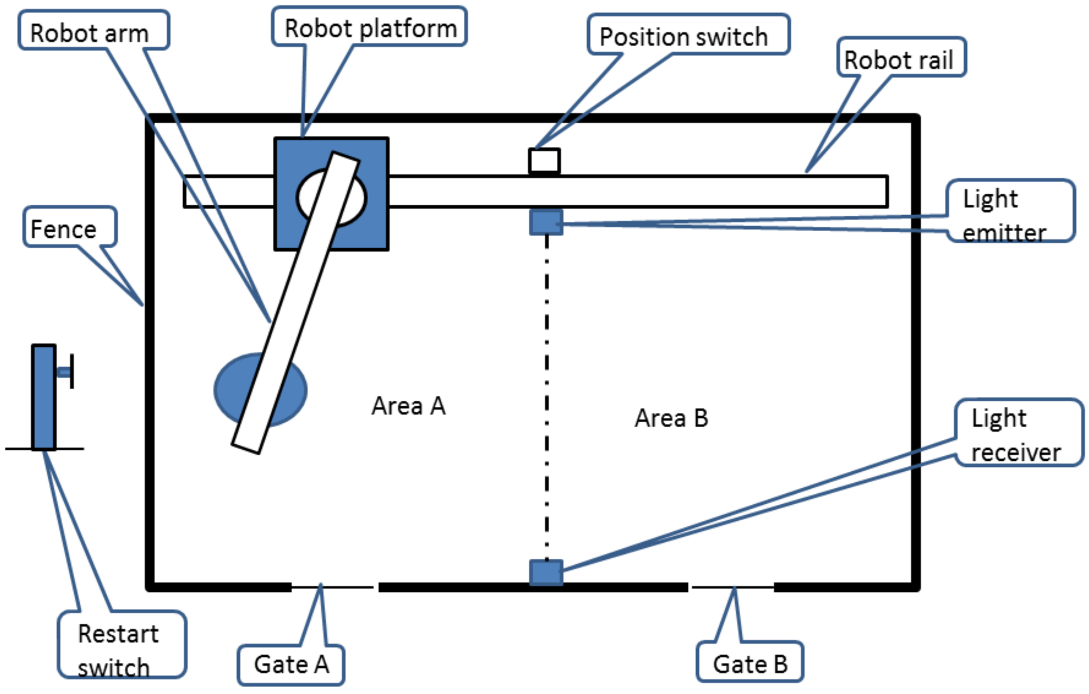
Krav 9	Inkonsekvent		Tvetydig	
Forklaring:				

Krav 10	Inkonsekvent		Tvetydig	
Forklaring:				

Krav 11	Inkonsekvent		Tvetydig	
Forklaring:				

A.3 Å skrive krav med boilerplates

I andre del av oppgavene skal du skrive noen krav ved å bruke boilerplates selv. Kravene er i denne delen relatert til eksempelet med en robotarmen som beveger seg på en skinne mellom to områder. Systemet ble kort presentert i innføringen før eksperimentet, men vi oppsummerer kort hva systemet gjør her:



“Robot platform” beveger seg frem og tilbake på skinnen kalt “Robot rail”, mellom de to områdene A og B. Robotarmen kan rotere på platformen, og utføre forskjellige handlinger med den andre enden av armen rundt om i området den befinner seg i.

Rommet er delt på midten, og to sensorer fanger opp bevegelse fra en side til en annen. Lyssensoren “Light receiver” detekterer om noen krysser den stiplede linjen hvis lysstrålen fra “Light emitter” brytes, og “Position switch” detekterer hver gang robotplatformen beveger seg fra et område til et annet.

I tillegg finnes det to innganger til rommet, “Gate A” og “Gate B”, som leder henholdsvis til område A og område B. Disse er også utstyrt med sensorer for å detektere om noen kommer inn i rommet.

Utenfor rommet finnes det en “Restart switch”, som starter roboten etter en stans.
Systemet som styrer roboten kalles “Robot control”.

Sikkerhet er veldig viktig i dette systemet, og den viktigste regelen er at roboten ikke skal operere i samme område som noen befinner seg. Du skal derfor, ved hjelp av boilerplates, skrive krav for å dekke de følgende tre reglene:

Oppgave 3

Hvis noen kommer inn i området der roboten opererer, enten gjennom en av inngangene, eller ved å krysse linjen mellom område A og B, så skal roboten stoppe innen 10 millisekunder.

Boilerplate requirement(s):

Oppgave 4

Hvis roboten flytter seg inn i et område der det befinner seg noen, så skal robot control stoppe roboten umiddelbart.

Boilerplate requirement(s):

Oppgave 5

Roboten skal ikke kunne startes hvis ikke begge inngangene er stengt og “reset switch” trykkes på.

Boilerplate requirement(s):

A.4 Evaluering

Vurder hver påstand under, og kryss av for alternativet som passer best.

	Veldig enig	Litt enig	Nøytral	Litt uenig	Veldig uenig
Jeg synes boilerplates var lett å lære					
Jeg synes boilerplates var vanskelig å lære					
Det var lettere å oppdage tvetydighet i krav skrevet med boilerplates enn i krav skrevet i fri tekst					
Det var lettere å oppdage inkonsistens i krav skrevet med boilerplates enn i krav skrevet i fri tekst					
Krav skrevet med boilerplates var vanskeligere å forstå enn krav skrevet i fri tekst					
Det var vanskeligere å skrive krav med boilerplates enn med fri tekst					
Jeg la merke til flere detaljer i krav skrevet med boilerplates enn krav skrevet i fri tekst					
Jeg så ikke forskjell på krav skrevet med boilerplates og krav skrevet i fri tekst					
Hvis mulig, vil jeg foretrekke å bruke boilerplates til å skrive skrive krav til et fremtidig prosjekt					
Hvis mulig, vil jeg unngå å bruke boilerplates til å skrive skrive krav til et fremtidig prosjekt					

A.5 Boilerplates

Boilerplates kan deles inn i prefix-, main- og suffix boilerplates. Alle krav skrevet med boilerplates må inneholde en "main boilerplate", som beskriver ønsket funksjonalitet. Kravet kan også ha en "prefix boilerplate" først, som beskriver enten en betingelse eller situasjon som skal utløse handlingen beskrevet i hoveddelen, eller målet med kravet. Til slutt kan kravet også ha en "suffix boilerplate" til slutt, som kan beskrive rekkefølge, unntak, tidsbegrensninger eller frekvensen funskjonaliteten skal forekomme med.

Prefix boilerplates

BP27: if <event>, ...
BP28: if <state>, ...
BP29: in order to <action> ...
BP30: in order to achieve <goal> ...
BP31: while <state> , ...

Main boilerplates

BP15: <system> may be <state>
BP16: <system> shall <action>
BP17: <system> shall allow <entity> to be <state>
BP18: <system> shall be <entity>
BP19: <system> shall be able to <action>
BP20: <system> shall have <entity>
BP21: <system> shall have <quality factor> of at least <quantity> <unit>
BP22: <system> shall have <quality factor> of at most <quantity> <unit>
BP23: <system> shall not <action>
BP24: <system> shall not allow <action> BP25: <system> shall not allow <entity> to <action>
BP26: <user> shall be able to <action>

Suffix boilerplates

BP0: ... after <event>
BP1: ... at <entity>
BP2: ... at least <quantity> times per <unit>
BP3: ... before <event>
BP4: ... during <state>
BP5: ... every <quantity> <unit>
BP6: ... except for <action>
BP7: ... for a period of at least <quantity> <unit>
BP8: ... from <entity>
BP9: ... other than <action>
BP10: ... to <entity>
BP11: ... unless <state>
BP12: ... within <quantity> <unit>
BP13: ... within <quantity> <unit> from <event>