Project Agreement Guidelines for Software Students

Abstract. Guidelines on writing project agreements in software development programmes, aiming at improving the quality of project definitions, and at making it easier to get them accepted. We primarily talk about final theses projects, devoting a section to smaller projects.

Disclaimer. This is *not a normative* document. I have no power to introduce regulations. It is an *information*, written from my point of view, as the programme head, after having consulted several teachers involved in the SDT programme and in the Bachelor in Software.

The study board has issued official requirements for projects, available at: http://www1.itu.dk/graphics/ITU-library/Intranet/Uddannelse/Eksamen/Project%20and%20thesis%20goal%20descriptions.pdf. They do not however give any guidelines for writing project descriptions. Use it to quality check the contents of your project.

1 Why ?

It is a task of a student group to write a project agreement, with some advise from the prospective supervisor. The agreement is officially registered in the project base. To access the project base log into http://my.itu.dk.

When accepted by both sides (the students and the advisor), and confirmed by the study board, the project agreement becomes the legal basis for the final exam. It is made available to the examiners together with a copy of your report. Report results are evaluated against the project agreement.

A messy project agreement can seriously impact your grade. A good definition will improve your exam experience and helps to start working on your project more smoothly.

2 Problem Definition

A thesis is an *academic project*. A typical academic project has the following components: a problem statement, a problem analysis, choice of a solution method, application of the solution method, and a reflection on the outcomes. Thus in order to define a project you *have to define the problem* that you want to solve. I suggest three simple steps to formulate a problem definition, approximately one paragraph per step.

1. *Introduction: What is the subject area?* This paragraph provides a very basic background on the problem. It serves as introduction for readers of the project agreement. Some examples of introductions from actual projects agreements:

> Code duplicates (clones) are a common problem in large code bases. They occur when a programmer reuses code through copy-and-paste instead of referencing the existing code. The reasons for doing so can be numerous. We just acknowledge that clones are likely to occur in large codebases.
>
> From a project titled *Refactoring of Dynamics NAV*

> Some Open Source communities consist of large numbers of individuals collaborating on the development of Open Source software products. These

> individuals are distributed, in every sense of the word. They collaborate across geographical locations, often without any geographical organizational center. They do not necessarily share educational backgrounds, professional experience level, personal maturity, working culture, work ethics or cultural values. Still many of these communities manage to build and maintain a collaborative modus and organizational culture that enables them to efficiently achieve the goals they set out for the project.
>
> Tor B. Sørensen. *Efficient open source communities. What drives them?*

> Feature modeling is the activity of modeling common and variable properties of products. A feature diagram consists of a set of nodes, a set of directed edges and a set of edge decorations. It is known that such feature models can be translated into propositional formulas.
>
> Yuan Gao. *Flexible Interactive Derivation of Feature Models*

2. *State of the Art.* This paragraph aims at narrowing the problem, by listing typical existing solutions, or research results in the field. For example:

> Within his thesis Michael Schou Christensen has developed an efficient way of finding duplicate source code matches in large codebases. Upon his framework, Tijs Slaats, Till Blume and Roland Schlosser have developed a specific implementation for the C/AL language.
>
> Tijs Slaats. *Visualization of code duplication matches in large codebases*

> In 2007 an algorithm for computing reduced feature models has been presented by Czarnecki and Wąsowski. This algorithm needs to be extended in order to construct a regular feature model.
>
> Victoria Kuzina. *Interactive Derivation of Feature Diagrams*

3. What is the *specific* problem *you* are solving? What is specific about your envisioned solution? Often this is easiest described as an improvement over existing work, described in step 2. Sometimes it is more convenient to swap these two points—define your task first, and only afterwards argue that this is an improvement. You can choose either order.

   This is the most important paragraph. Make sure that you use *verbs* that clearly state what you will do. Examples:

> During their project they [Slaats et al] noted a number of possible improvements and enhancements to Michael's framework, one of these being the visualization of the code duplication matches that were found. The problem they found with the current representation of matches is that it can be hard to get an overview of how the matches relate to each other, especially for people not already familiar with the original base.
>
> The focus of this project will be to look into this problem and find more robust visualization techniques that can be used to represent these code duplication matches.
>
> Tijs Slaats. *Visualization of code duplication matches in large codebases*

> This thesis will outline the basic of collaborative moduls and organizational culture and attempt to identify the main factors contributing to the establishment and maintenance of these tenets. In other words the research question

> of this thesis paper will be: "What are the main contributing factors that build and maintain an efficient collaborative culture in large Open Source Communities?
>
> Tor B. Sørensen. *Efficient open source communities. What drives them?*

In more generic terms examples of suitable problem definitions are:

- Implement an algorithm and evaluate it experimentally on a specific class of inputs; compare with competing solutions.
- Read a book that introduces a certain method (let us say programming paradigm) and apply it to solve a specific problem. Reflect on the benefits of using this method in this particular case.

Examples of projects that do not normally qualify as research (and thus risk rejection):

- Implement an algorithm.
- Read a book.

It is important to see that the two latter project kinds can in principle have the same contents, as those specified formerly. Nevertheless the former project descriptions are acceptable, whereas the latter are not. They lack detail in description, that warrants academic level of activities. The first one lacks reflection and analysis, the second one lacks any intellectual creativity.

Agreements are often rejected not because their prospective content is not at the right level, but because the academic activities are not explicit in the description.

Finally remember to be explicit about the size of the task, if this is not visible from the problem definition. For example, instead of saying "implementing several example applications in our LISP dialect", say "implement several example applications up to 200 lines of code in our LISP dialect". This way you leave less space for interpretation at the exam, and avoid mismatch between your own expectations and your supervisor's expectations.

3 Method

Describe how you are going to address the problem defined above. In here you should include both *general* and *specific* methodological plans.

General methodology: What are the basic tools of your work? User studies? Experiments? Prototyping? Mathematical proofs? Literature surveys? How are you going to collect data (if any)? How will you evaluate quality of your design or implementation?

Specific method: Apart from general methodology, indicate specific methods, if you are planning to apply them. Examples are: model driven engineering, object oriented analysis and design, proofs with logical relations, original research methods/processes outlined in a book or a paper that you plan to follow, etc.

4 A Checklist

- You must be able to report about your findings. If you cannot write a clear report based on your work, then this is not a good project definition.
- Make sure that you have described the objectives precisely and unambiguously, so that the examiner can assess whether you have met your goals in the end. If your definition is vague or hard to understand, you risk being graded against an interpretation of you description, not the description itself.
- Check the spelling and grammar in the title of the project. This is exceptionally important, as this title would usually end up in your transcript of records when you graduate from ITU.
- A short bibliography is welcome, especially in definition of thesis projects. In most of the projects it is easy to point out to the main reading on the problem definition, and to the main reading about the solution method.
- Spell check the entire text. Ask a friend to proofread it.
- The project agreement has to be understandable to non-experts in the field. Run it through your fellow student at ITU, who does not focus on the same narrow area as you do. Ask him whether the description is understandable.

5 When?

Do not plan to send your project proposals to the prospective supervisor in the very last moment before the deadline. The project proposal has to be accepted by your supervisor, and this often requires several cycles of discussion and improvements.

By sending the project for approval just before the deadline you put your advisor in an uncomfortable position of choosing between accepting a badly written proposal, which will cause trouble later, or rejecting the proposal right away, causing trouble for you immediately on the spot.

6 Smaller Projects

Of course a smaller project (7.5–15 ECTS) can also be devoted to research, and you can write its definition following the above advice.

However, for many smaller projects the goals can well be less ambitious. You can for example spend the project time on learning a specific technology. Then it is important that you specify explicit learning outcomes, which can be tested at the exam (see in the course base, how the IT University specifies learning outcomes for your courses). You should definitely ask your supervisor to help you with outlining these.

A rule of thumb: state what you will be able to do after the project, not what you will know.