DD1392 MVK-14

Software Engineering Project

Student Handbook

Author: Karl Meinke (karlm@nada.kth.se)

Version 1.31

2014-01-13

Abstract. The course "Software Engineering (MVK)" has associated practical project work, which contributes 9 points to the overall course credits. This handbook aims to provide the most important practical project information to every student in one single document.

Contents.

1. Overview.

The course DD1392 aims to introduce students to the theory and practice of software engineering via a large group project. Groups involve a number of students and 1 external project provider. It is **compulsory** for students to finish the group project. No points can be awarded for students who fail to complete the project.

This document summarizes the activities needed for each deliverable of the software project, from *initial planning* to *final software demonstration*. It can be viewed as a "*one-stop-shop*" for all practical details concerning projects. However, **it should be studied together with the course notes and marking guidelines**, which cover the theoretical software engineering ideas and techniques needed to execute the project and produce the documentation on which the course grade is mainly based.

2. Course Grading.

The course gives 9hp.

Performance on the course for individuals on the Bologna scale (A-F) will be assessed by the grades (U, G, VG) awarded for the six project deliverables (1) PPD-1, (2) PPD-2 (3) URD-1, (4) URD-2 (5) ADD-1 and (6) final software demonstration (DEMO). Each deliverable can be awarded the grade VG (2 points), G (1 point) or U (0 points). **Rule 1: Any deliverable which is graded U must be re-submitted until it is passed, and does not qualify for the grade VG on later re-submissions.**

The final course grade formula is linear:

Final grade = 2 * PPD-1 + 2 * URD-1 + 2 * ADD-1 + 1 * PPD-2 + 1 * URD-2 + 1 * DEMO

This formula gives a maximum of 18 points. The grade intervals are:

A = 18 points

B = 17-16 points

C = 15-14 points

D = 13-9 points

E = 8-3 points

The grades FX and F cannot be final grades according to Rule 1, since you have not completed the course.

Note also: **Rule 2: It is not possible to get a VG for any late deliverable**. We consider timely delivery to be an essential part of every engineering discipline.

The grade returned for each deliverable is shared equally by all project members, unless there is reason not to (e.g. *non-participation*). Minutes of meetings will be used to determine non-participation, so it is important that you attend group meetings. There is no formal examination.

3. Project Activities

This chapter describes the four different kinds of activities that you should be engaged in during your project. Each activity leads to a different project deliverable, which is either a document or piece of software. The 4 activities are:

(i) *Project planning*,

(ii) *Requirements Capture* and *Analysis,*

(iii) *Architectural* and *Detailed Design,*

(iv) *Coding, Testing* and *Delivery.*

The first three activities lead to documents (PPD, URD and ADD). The final activity leads to a software deliverable (DEMO).

The order that you perform activities in will be up to your group to decide. You may even perform some activities *in parallel* (e.g. by dividing up your group). There are a variety of software development lifecycle models that will be studied in the course. You should choose the most appropriate.

However, there *are* certain natural prerequisites:

(i) it is difficult to run any project without some initial plan (PPD),

(ii) it is difficult to begin design work without any requirements (URD),

(iii) it is difficult to start coding without any design (ADD),

(iv) it is difficult to do testing and delivery before coding. (DEMO)

So the deliverables of each activity should drive the following activity. Nevertheless, later activities can also drive earlier activities, and this is known as iterative development.

You may find that each deliverable needs revision at some later stage. This is often difficult to foresee. Nevertheless you must be careful not to let the project *over-run* for lack of planning or bad planning, these mistakes are painfully common, but will lose you points (see Section 2)!

Next we will look at each of the above four activities, and their deliverables.

3.1. Activity: Project Planning.

3.1.1. Overview.

Initially, project planning work is meant to give you some time at the start of the course in order to:

(i) *set up* your group,

(ii) *choose* the right project for your group,

(iii) *orientate* yourselves about your chosen project and its context and

(iv) *think* critically about how your final product might look by *surveying* existing products which are similar.

Later project planning may have to adapt to significant changes in the project, such as user requirement or delivery changes, or loss of critical project members.

Therefore, you must keep you project plan updated at all times, using careful version control and change management.

The deliverables for all project planning activity are the *project planning documents* (PPD-1 and PPD-2).

3.1.2. Timetable

Week 4, Mon 20 Jan 2014, 1500-1700: Introduction to projects and choice of projects.

Week 4, Thu 23 Jan 2014, 1200: **Deadline**: Each project leader to **e-mail** the ***project number*** and ***project title***, for their group's **top three project choices** to Karl Meinke (karlm@csc.kth.se). Note: groups generally get the first or second choice on average.

Week 5, Mon 27 Jan 2014, 1500-1700: Announcement of project allocation in class.

Week 8, Wed 19 Feb 2014, 1300-1500: **Group project presentation**: main focus is delivery of PPD-1.

Period 4 March 2014: **Group project presentation**: main focus is delivery of PPD-2 and also ADD-1.

3.1.3. Initial Goals In order that your project progresses smoothly, it is important that you function "as a group". This may involve getting to know one another as students (perhaps with different skills, interests and experiences) as well as establishing consensus on how to discuss, reach agreement and resolve any disagreements. These are all important "political" aspects of any engineering project. You will need considerable "*team spirit*" to get through the hard parts of your project.

Project planning activities, if carried out successfully, will establish you as a coherent group, with an agreed and understood joint set of objectives and approaches. This will

immediately lower the level of *project risk* attached to your project. (A more formal *risk analysis* comes later.) Furthermore, they should help you prepare for your first major task: *user requirements capture and analysis*.

3.1.4. Activities As soon as possible, you must meet as a group and establish a regular weekly meeting time that is convenient for everybody. You can meet in any location you wish. **Unfortunately, we cannot take responsibility for booking (and unbooking!) meeting rooms for you.** Your meetings will be unsupervised. It is suggested that from time to time smaller working groups can also meet. However, to maintain progress you must meet every week. Evidence of this meeting, that must be submitted, will be a set of printed minutes for each meeting. **You must submit a full set of minutes of each project meeting as an appendix to your PPD.** Minutes of meetings should always be placed in your online project workplace,

(i) to record what was agreed, and

(ii) for the benefit of any members who were absent because of illness etc.

At the start of every group meeting you should go through the previous meeting's minutes, to check whether planned actions have been successfully implemented or not.

3.1.5. Project Membership. Your first task is to assign yourselves a group name (legal and decent!), and collect your names, personal numbers, e-mail addresses and telephone numbers. You should write these up in a **Project Member Document (PMD**), place this on the project web site (3.1.6), and send one copy by e-mail to your project examiner.

3.1.6. Project Web Site. You must set up a web site for project collaboration that allows you to share project information, minutes, code, etc.

3.1.7. Skills Audit. You must perform an audit of the skills available in the group, and the technical strengths (and weaknesses!) of each of each group member (e.g. *Unix expert, Windows expert, graphics expert, industry experience*, etc.) Team members need to be honest and open about this to avoid unpleasant surprises later.

3.1.8. Project Staffing. Using your skills audit (3.1.7) you will create your organization. On the basis of skills, interests and personalities you **must** assign the following roles to at exactly one group member

Project leader Responsible for overall co-ordination, and ultimate decisions and responsibility, deep understanding of the project and the PSS 05 documentation standard.

Project secretary Responsible for writing/delegating documentation and report writing, taking minutes of meetings, deep understanding of the PSS 05 documentation standard.

Customer Account Manager Industrial project providers tend to be very busy, so one single person who is a (reliable!) unique point of contact often reduces confusion and makes communication work best. Excellent social skills are important for this role. It does not suit a shy

or very introverted person.

You **may** also assign some of the following roles to group members. Note that any member can have more than one role at any time.

Chief programmer An optional role, but a model favored by many in industry. Has senior responsibility for coding. Usually an expert, who also delegates simpler tasks to less experienced programmers.

Programmer A more flexible programming role that might also include documentation and testing responsibilities.

Documentation Manager An optional role that could be separated from the secretary role. A person who writes and/or produces reports.

Technical Writer A deliverable writer. Usually assigned tasks by the secretary or documentation manager.

Tester Responsible for unit, integration and system testing. Receives testing tasks assigned by the project leader or chief programmer. Typically programmers should not be allowed to test their own code.

Requirements Analyst. Responsible for the capture, exploration and analysis of end-user requirements. Often uses an iterative process, with the help of rapid prototypes including GUI designs and mock-ups, interviews, focus groups, etc. Good communication skills are important for this role, as well as technical ability.

System Designer. Responsible for problem analysis, solution inception and software design at different stages in the project. Design tasks are normally assigned by the project leader.

Human Interaction/GUI expert. A common specialist role that requires particular training and experience. May involve cognitive psychology, graphics design, communication theory and even copywriting. Tasks assigned by project leader/ chief programmer.

Project planner. A role that could also be performed by the project leader and/or secretary. The project planner should estimate workloads, establish schedules, monitor progress and reschedule delayed tasks to incorporate change. Realism and accuracy are important.

These roles should be recorded in the PPD. Although you may have specific technical roles, everyone is expected to contribute to the project ideas and decision making. Friction in the group may be reduced by being democratic, although **in the case of an unresolved dispute the project leader must have the final say.** The *project provider* may also be able to resolve certain disputes for you.

3.1.9. Background Research. After you have established your group, you will need to take contact with your *project provider* and investigate the background to your project

task. This particular project planning activity is intended as a precursor to user requirements capture.

Your goal for background research is to produce an initial PPD draft according to the prescribed template below. If you produce a good initial PPD your task in capturing user requirements will be much easier, so it is an investment in your course grade!

3.1.10. Deliverable:

Project Planning Document **(PPD-1 and PPD-2). (2 printed hard copies, 1 electronic copy e-mailed to karlm@csc.kth.se)** The PPD describes your initial project management decisions. It summarizes your background research into the problem, culminating in a feasibility assessment. It also identifies any learning objectives necessary to carry out the project. The PPD follows the documentation standard defined in the template below.

Deadline: End of course:

PPD Template (Outside the PSS-05 standard)

(Actual template headings and components are in **bold** `Courier font`.)

a - Title page,

1. **`Project title`,**

2. **`Project Group Name`,**

3. **`Project Members`** all members in alphabetical order by family name.

4. **`Version number and document status`** e.g. working draft, final draft, revised version. (I suggest version numbers: 0.x = preliminary draft, 1.0 = submitted final draft, 1.x = subsequent changes, 2.x revised URD.

5. **`Date of printing`,**

b – `Abstract` 1-2 paragraphs.

c - `Table of contents`

d - `Document Change Record`, A table describing the major changes made for each document version.

1. Statement of the Problem. You must describe the problem chosen.

1.1. Detailed Statement of the Problem. A description of the problem, **in significantly greater detail than the description handed out in the project list**. You must provide your own thoughts and interpretation of the problem. This section should include all the new information you have obtained by talking to your project provider.

1.2. Motivation. Explain why you chose this project. Why is it important for the group? Does it enhance job skills. Does it satisfy some interests?

1.3. Goals. A reflective analysis of your own project choice. What do you hope to achieve with this project choice in terms of: (a) personal goals, (b) career goals, (b) educationally goals, i.e. what will you learn?

1.4. Skills Baseline. What skills do you have in your group? Are there individuals with specialist skills? Will you need to learn any new skills? How will you learn these?

2. Background to the Problem. This section should be a study of examples of the type of system that you intend to build. Your study will be based on a literature survey: (books, magazines, trade newspapers) and a web search. One aim of this study is to compile a list of possible product features that can be used to steer Phase 2, the user requirements phase. You should give extensive *references* to all sources of information that you use, such as books, URLs etc.

2.1. Commercial background. (*Appropriate for a commercial or "saleable" product including a prototype*.) What commercial products exist similar to your own? (You can include freeware, shareware, web applications etc.). Who makes them? What do they cost? Who buys them? Are they for computer experts or beginners? Are they national or international? What skills does a typical user have? Is the market mature or immature? Again, you should give extensive *references* to all sources of information that you use, such as books, URLs etc.

What is the "best available product" of this type. Why is it best? What features make it the best? You may be able to consult industry analyst reports produced by such companies as Gartner Group. How many of these features could you realistically implement in your own product given the time available? What features could be classified as (a) economy (or student) version, (b) standard version, (c) deluxe (or enterprise) version.

Expect to use a web search, magazines and trade newspapers extensively here.

2.2. Scientific background. (*Appropriate for a research or experimental application*.) If you have a research-oriented project there may be little or no commercial literature available. This might be the case if your project provider is a teacher or researcher at KTH. In this case you will need to use course notes, research journals, conference proceedings and library books extensively.

What algorithms and data structures might be needed for the project? What research subjects are related to this project? What are the minimum number of features needed to make a viable tool or demonstrator? What could a commercial product look like in the distant future? Are there research experts at KTH or elsewhere in this subject? Can you contact them for information?

Can you classify features according to an (a) economy (or student) version, (b) standard version, (c) deluxe (or enterprise) version. If not, how else could you classify them?

2.3. Technical Background. In this section you should estimate what platform(s), operating system(s) and programming language(s) would be appropriate. Can you use any commercial-off-the-shelf (COTS) products to solve parts of the problem? e.g. a commercial database, spread-sheet front end, web browser interface etc. What licenses for COTS products are available at KTH and what licenses will your end user provide? Can you use application generator tools, e.g. *compiler-compilers* like Yacc, Lex, Javacc or *component libraries* e.g. Swing?

3. Risk Assessment.

You should try to identify as early as possible the most significant risks to your project. These could be technical, customer or staff related. This initial assessment will later be extended to a more detailed and systematic risk analysis. *In the first release of the PPD, without formal training, you should assess risks in whatever way you understand them.* Later, in the final release of your PPD, you must use the results of a systematic risk analysis template to describe the project risk situation.

From your risk assessment, you must identify the top *n* risks going forward, and for each risk, you must give a risk management proposal. (In the first draft *n* could be 2-3, in the final draft, *n* should be at least 7.) Note that the risk assessment document itself can be included as an *appendix* at the end of the PPD.

4. Project Plan.

Estimate a schedule for the project. Include major activities (how long), milestones (when), and deliverables (what). Deliverables at various milestones should include: *document drafts, software prototypes, designs, software components, customer deliveries, final documents, etc*. Analyze the effects of *project slippage*.

Estimate the effort required for each deliverable. Schedule time and people for each deliverable. Produce a task allocation and a personal schedule for each project member for the remainder of the project. This information could be presented in a table. The project manager needs to check that each project member is producing deliverables according to their personal plan.

`5. Feasibility Assessment.` On the basis of what you know, you need to draw conclusions about the feasibility of your project, based on the time and work force available, the level of ambition of your project (based on project features), and the skill set available in your group.

`References:` including URLs, books, journals, conferences, newspapers etc.

`Appendix.` The PPD must contain an Appendix publishing the list of all group members. For each member, the Appendix must list all the job roles that member has in the project. This list can be modified later according to need.

3.2. Activity: User Requirements Capture and Analysis.

3.2.1. Overview. User requirements capture and analysis concerns a definition of *functional* and *non-functional requirements* on the application from the user's perspective. Usually we try to just describe the *problem* without identifying any *particular solution*. For commercial clients, the technical problem may be embedded in a bigger problem of the client's *business model*. In this case, it may be necessary to understand this wider business model.

Often, solution constraints have to be taken into account from the beginning, such as hardware platforms (e.g. mobile phones), operating systems, programming languages, existing IT infrastructure etc. It is even possible that prototype solutions are needed early to help define the problem! Capturing user requirements usually also involves *data modeling*.

3.2.2. Schedule.

Week 10, 9 Wed 5 Mar, 13.00-15.00, E1: **Group project presentation**: main focus is delivery of URD-1 and risk analysis results.

Period 4, May 2014: **Group project presentation**: final project presentation includes URD-2 delivery.

3.2.3. Goals Your goal is to capture and analyze user requirements, starting from an initial exploration of product ideas that you carried out in your background research (see Section 3.1.9)

You will have to interact intensively with your end users (normally found within your project provider company, but possibly also including customers of that company itself). You will need to hold customer meetings in which your record details of user requirements. After each such meeting you should (1) review the output of the meeting, (2) write up the requirements carefully and discuss them, and (3) take the output of these discussions back again to the company to confirm that your recorded requirements are complete and correct. If not then you will need to modify them and iterate the process. The whole activity may take many iterations, and be extended over almost the entire project, depending upon your circumstances.

3.2.4. Activities.

(1). You need to choose one or more requirements capture techniques (such as focus groups, brainstorming, questionnaires, task demos, document studies, etc. see my lecture notes!)

(2). You should carefully book ahead and plan every meeting to optimize your use of your client's time.

(3) You should carefully choose who participates in client meetings from your own group. You should also allocate tasks to non-participants, such as requirements documentation, minutes, project web-site management and requirements analysis. Explaining captured requirements to the non-participant members of your group after each meeting is an excellent way to clarify and debug them!

(4) You should carry out an appropriate number of client meetings to at least achieve a (more or less) *stable initial set of requirements*. You should flag all requirements that seem inherently *unstable* and liable to change before the end of the project.

(5) You should analyze requirements for *validity*, *completeness*, *consistency, technical feasibility* and *verifiability*. Your client may suggest ideas for verifiability including *acceptance tests*.

(6) You must produce a **data model**, either as an *object diagram*, *class diagram*, *ER diagram*, or a *data dictionary*.

(7) I recommend you produce an early GUI design, at least as a paper design. You can take this further by using visual programming tools to produce a simple prototype. You can use any language here (even html!), since the work could be thrown away and rebuilt later in your target language. You should find that the GUI design helps requirements

capture, e.g. with a task demo, the user can explain how they would expect to use the software.

(8) You should systematically convert all requirements into the PSS-05 format, (table format in Section 3) and attempt to **organise** the requirement information, e.g.

1. as a *hierarchy* based on increasing detail (i.e. *refinement*),
2. using component relations (UML *is_a* and *has_a* or *functional dependency*)

This organization needs to support *navigation*, and and *overall view*. Non-functional requirements cannot always be organized in a hierarchy, so try to find other natural ways.

(9) When you have completed an initial requirements capture and analysis that would allow you start architectural design work, you are ready to produce the *first draft* of your URD document. For this, you **must** follow the PSS-05 format exactly. If you are unsure about document structure you should contact me. This first draft must be *regularly maintained* and *revised* until it becomes the *final draft* that you submit for grading.

(10) Think about what information can be extracted to give a short presentation of the document (e.g. diagrams, GUI designs, data models, etc.) Generally it is better to present visual information than text.

3.2.5. Deliverables

There are **two deliverables.**

User Requirements Document **(URD-1 and URD-2). (2 printed hard copies, 1 electronic copy e-mailed to karlm@csc.kth.se)** The URD describes the user requirements on the system that you are to build, including interfaces to any existing software/hardware systems that must be integrated with your product. The URD follows the PSS-05 documentation standard defined in the template below.

PSS-05 User Requirements Document Template (URD)

N.B. In all PSS-05 document templates, if there is no project information relevant to a section, the following should appear below the section heading: `*This section not applicable'*, possibly with the appropriate reasons for this exclusion.

My comments on the tables of contents are enclosed in parentheses. Section titles which need to be provided by document authors are enclosed in square brackets.

Every PSS document must contain the following *service information*:

a - Title page,

1. **Project title,**

2. **Project Group Name,**

3. **Project Members** all members in alphabetical order by family name.

4. **Version number and document status** e.g. working draft, final draft, revised version. (I suggest version numbers: 0.x = preliminary draft, 1.0 = submitted final draft, 1.x = subsequent changes, 2.x revised URD.

5. **Date of printing,**

b – Abstract 1-2 paragraphs.

c - Table of contents

d - Document Change Record, history of changes made at each issue.

1 Introduction. This chapter should give an overview of the whole document

1.1 Purpose. The purpose of this document and it's target audience. Convey the intention of the document to clients as well as project members, e.g. is it the basis of a legal contract or informal agreement? What aspects of the document are binding?

1.2 Scope of the software. An "*executive summary*" of the product under development. Not more than 30-40 words.

1.3 Definitions, acronyms and abbreviations. Especially technical terms and acronyms (XML, ASP, TTCN UML, CORBA, etc etc) unfamilar to the reader and/or customer.

1.4 References. Sources of additional information helpful in reading this document, with a brief explanation of the contents and usefulness of each. Could be customers in-house reports, reports from previous projects, scientific or technical reports, industry white papers, computer science or other books, on-line references (URLs) and related web sites, newspaper articles.

1.5 Overview of the document. Provides a birds-eye view of what information is given in this report, and where in the report it can be found. Description can be focused towards different types of reader, e.g. end-user, technical, developer, specialist, domain expert, accountant, legal, management, customers customer etc.

2 General Description.

2.1 Product Perspective. Describes related external systems and subsystems. How the product under development fits into an existing environment. Business oriented account of product need, including relevant business model information. Business case for developing the application.

2.2 General Capabilities. Describes the main capabilities required and why they are needed from the end users perspective. Gives an overview of the product under development. Takes a user-centric approach.

2.3 General constraints. Describes the main constraints that apply to the product, and why they exist. Describes the constraints on data in the form of a data model, which may be an object diagram, class diagram or data dictionary. (Note: a data model must be presented.) Includes limitations on functionality due to limited project scope (time, personnel, money) and limited customer needs (ambition, money, skill level). Also includes any technological and scientific constraints, e.g. performance, bandwidth, computational difficulty of problem solving, lack of efficient algorithms, etc.

2.4 User characteristics. Describes who will use the software, expected background, previous training and level of skill (may be several). Identify different job roles and contexts of use (can be used to develop a use case analysis). Used to determine user interface requirements, online/offline user support and product documentation.

Based on user characteristics, present a usability analysis that clearly identifies technical requirements on the user interface structure. This might include use case information to describe important workflows. If appropriate, suggest a suitable interface metaphor (e.g. desktop, capture/replay, spreadsheet, etc.)

2.5 Assumptions and dependencies. Describes the assumptions upon which the requirements depend. For example technical assumptions on levels of hardware performance, system availability and reliability. May include commercial assumptions about customers business needs/ business model/ business process.

2.6 Operational environment. Describes what any external systems do, in terms of functionality, (e.g. file servers, databases, etc). Describes the interfaces made available to the product under development. (Technology oriented).

3 Specific Requirements. Provides a detailed list of specific requirements organized into two categories.

3.1 Capability requirements. Functional requirements on the system, (what the system should actually do) including interface requirements (e.g. file formats, data definitions, APIs, communications protocols etc.) in so far as these are known at an early stage.

3.2 Constraint requirements. Non-functional requirements, including:

- *performance requirements* (speed of execution, memory requirements),
- *environment requirements* (hardware, OS, peripherals, network, web browser)
- *external requirements* (minimum version numbers of external systems, subsystems, functionality needed from these)
- *reliability requirements* (uptime, mean time to failure, accessibility, loading, average performance, worst case performance, etc)
- *usability requirements* (minimum time to learn system, expected time to learn system, level of user support, expected efficiency gains from system)
- *safety requirements* (critical functionality and its reliability)
- *legal requirements* (conformance to industry standards and recommendations)

`3.x.y.(z) Individual Requirement Template` (x = 1,2)

`Identifier`	Symbolic identifier using helpful abbreviation.
`Requirement Description`	A structured description that includes symbolic references to related requirements to aid explanation.
`Justification`	An optional justification (or motivation) of why this requirement is needed.
`Need`	Use a scheme such as Standard/Economy or Deluxe/Standard/Economy.
`Priority`	High priority means early delivery needed, low means late delivery is acceptable.
`Stability`	Unstable requirements that can change must be flagged.
`Source`	Origin of requirement. Could be an individual group member, whole group or external. Traceability information.
`Verifiability`	Explains how to verify requirement. Each requirement must be verifiable. It must be possible to: (1) check requirement is in the design, (2) test that the software does implement the requirement.

3.3. Activity: Architectural and Detailed Design.

3.3.1. Overview.

Architectural design work concerns the construction of an architectural model that:

(i) meets the software requirements, including both functional and non-functional requirements and constraints,

(ii) uses an appropriate system decomposition into high and low level components. Software engineering principles should be followed, such as *high cohesion* and *low coupling* that will support long-term maintenance of the product

(iii) re-uses well known high-level architectures and low-level design patterns, to improve engineering quality, readability, reliability, maintainability and component re-use,

(iv) achieves any appropriate architectural tradeoff of resources (e.g. speed, memory usage, throughput etc)

(iv) partitions the final programming task among individual programmers in a way that maximizes independent activity (including code design, programming and testing) through appropriate modular design with clear and precise interfaces between components.

(vi) documents the design decisions clearly.

3.3.2. Schedule.

March 2014: **Group project presentation**: main focus is on delivery of PPD-2 and ADD-1, but also current status of the URD, and an update on risk analysis.

3.3.3. Goals.

(i) You will need to organize project meetings to carry out this work, and document the decisions of each meeting. When design decisions become stable you should use them to write sections of the ADD.

(ii) The purpose of each meeting is to make progress in identifying a set of high, medium and low-level system components that can implement the functional requirements described in the URD.

(iii) You will need to: (i) choose a high level architecture, and (ii) partition and assign medium and low-level components (such as software patterns, COTS components etc) to the system architecture components such as specific clients, specific servers etc.

(iv) A good choice of architectural design should maximize *cohesion* and minimize *coupling*, and so simplify future software redesign, fault repair and feature upgrade.

3.3.4. Activities.

(1). You will need to familiarize yourself with relevant standardized software architectures, such as those described in the lecture notes. You may also need to do research into other relevant architectural models.

(2) You should try to share and if possible extend your group knowledge of any relevant software design patterns.

(3) Based on (1) and (2) you should brainstorm ideas for a suitable architecture. Your URD might suggest several possible architectures.

(4) The performance requirements of your URD might suggest partitions of the architecture that raise performance of the system to the minimum required level.

3.3.5. Deliverable.

There is **one deliverable.**

Architectural Design Document **(ADD-1). (2 printed hard copies, 1 electronic copy e-mailed to karlm@csc.kth.se)** The ADD describes the architectural design of the system that you are to build, including interfaces to any existing software/hardware systems that must be integrated with your product. The ADD follows the PSS-05 documentation standard defined in the template below.

Deadline: End of course:

PSS-05 ADD Template

`1. Introduction.`

Similar to URD Section 1, but set in the context of this ADD report.

`1.1. Purpose.` See URD Section 1.1.

`1.2. Scope of the software.` May be revised from URD Section 1.2 in the light of feasibility studies, architectural analysis and design, research and development work, etc.

`1.3. Definitions, acronyms and abbreviations.` May extend/delete information from URD Section 1.3.

`1.4. References.` May extend/delete information from URD Section 1.4.

`1.5. Overview of the document.` Similar to URD Section 1.5, but describes the ADD. Again it need not be assumed that readership on the customer side exists. In

practise, this document may again be company confidential to the development team.

2 System Overview. Summarises: (i) the system context (how it fits into an existing framework of other packages and systems), and (ii) the system design. More detailed descriptions of (i) and (ii) are given in Sections 3 and 4 below.

3 System Context. Gives a detailed description of the system context, with relevant diagrams. Defines the external interfaces of the product under development to these other systems.

3.n External interface definition. Provides an interface definition to each separate external component type or physical component.

4. System Design. Provides an overview of the design techniques used, especially any in-house or non-standard methods, project specific methods, or non-standard interpretation of standard languages/methods such as UML.

4.1. Design method. Describes and references the design method used to construct the architectural model. Includes any methodologies such as RUP, SSADM, etc.

4.2 Decomposition description. Gives the top level view of the systems design, preferably with diagrams. Shows the major components which will be described in detail in Section 5. Identifies control and data flow between components.

5. Component Description.

Gives detailed component information according to a fixed template. Components may be top level components, identified in Section 4.2, or subcomponents of these. Preferably use a component identification scheme which is easy to read/follow/remember.

5.n. [Component identifier] Fill in name here.

5.n.1. Type. Could be a module, an input/output/temporary file, a program, a class, a script, a web page, etc.

5.n.2. Purpose. Describe the purpose of the component, and relate this to a numbered software requirement in the URD.

5.n.3. Function. Describe the functionality of the component, including its interface properties (call and return types) and logical behavior.

5.n.4. Subordinates. List the immediate subcomponents of the component, using defined component identifiers.

5.n.5. Dependencies. Describe the logical preconditions for using this component, e.g. files and/or objects that must exist.

5.n.6. Interfaces. Define the control and data flow to and from the object. Gives a detailed picture of its context in the overall system architecture.

5.n.7. Resources. List any resources required by the component, such as external components external subsystems, hardware, etc.

5.n.8. References. Reference any external documents needed to understand the component.

5.n.9. Processing. Describe the control and data flow between immediate subcomponents of this component. If the component has no immediate subcomponents (i.e. it is fully decomposed) then outline the method of processing used by the component to perform its task (e.g. with pseudo-code, state diagrams, etc).

5.n.10. Data. Describe in detail (where possible) the local data values and data structures belonging to (local in scope) this component. Otherwise give an outline description.

6. Feasibility and Resource Estimates. Summarise the conclusions of a feasibility study around the architectural model. Prioritise all components with a priority model (e.g. economy, standard, deluxe versions).

7. User Requirements vs Components Traceability Matrix.

Gives a table that cross references architectural components (based on defined component identifiers) to user requirements numbered in the URD. It may be appropriate to omit references to non-functional user requirements that are not code related (e.g. legal requirements).

Software requirements

	UR1	UR2	UR3	UR4	UR5
AR1	x	x			
AR2			x		
AR3			x		
AR4		x			
AR5	x				
AR6				x	
AR7					x

Architectural

requirements

4. Project Software Demonstration

4.1. Overview.

To complete your project you must hold a private software demonstration (DEMO) with your project examiner. Your demo will normally last about 1 hour. It should be held either:

in a CSC computer lab

in the examiners KTH office

We do not visit external sites for demos. If you have specialist hardware, you must make sure that it can run in one of these locations. The whole group does not need to attend. No one else will be present (e.g. other students) unless invited.

The main idea is to assess:

how well the overall product vision and detailed user requirements have been

fulfilled,

how and why you have deviated from project plans,

the quality of the final product.

Examiners do not normally inspect code unless there is a particular reason to do so.

They do go through the submitted URD, looking at each requirement, and assess how it has been fulfilled in the product.

They also try to judge product quality factors such as

stability (crashes),

performance (speed, throughput, memory usage etc,)

quality of the user experience,

quality of online documentation, …

4.2. Schedule.

Period 4, May 2014: **Group project presentation**: main focus is the final status of the project and review of all deliverables (PPD, URD, ADD, software).

You should make a private appointment for the software demonstration (DEMO) directly with your examiner, (and not through the course leader). You can do this by e-mail.

You should arrange a date for a demonstration once your software is considered complete and you are satisfied with the outcome of all testing. You can arrange a demonstration date before your final project presentation in class or after.

It is strongly advised that you choose a date well before the end of term if you wish to have your Ladok points credited before the summer vacation.

4.3. Goals.

Each group should construct a *software demonstration plan*. Your plan should decide:

a. how to use about 60 minutes of time,
b. who should speak,
c. what they will say.

You should save at least 20 minutes at the end for general questions about the product and the project. You should choose 1-2 competent speakers to present the product, but

no "sales talk" please …

it might be useful to have 1-2 senior programmers present to answer technical questions

It will be very helpful if you have written feedback from the end-user that the examiner can read.

The speakers should talk through a structured demo of the product, including some typical use cases. Try an "end-to-end" demo from startup through typical usage through to application shutdown. Try to demonstrate error handling capacity and robustness.Try to use realistic cases and examples, which may be prepared in advance.

Make sure your software is up and running before the examiner arrives (so your valuable demo time is not wasted). Trouble with startup will otherwise be considered as a system crash!

If you have not tested your system running in the CSC computing environment you must do so before the demo!

A **possible schedule** could be:

(i) 15-30 minutes structured demo

(ii) 10 minutes examiner tries out system on their own (if appropriate and they wish to do so)

(iii) 20 minutes question and answer, e.g.

Any discovered problems/bugs

Limitations and missing requirements

Best and worst features

Project execution and experiences

If things have gone well then the demo is normally an enjoyable experience, and you can be proud of your result. **With a well-planned software demo it is not necessary to be nervous!**

4.4. Final Project Presentation.

The final project presentation is a short time slot (5 minutes plus questions) in class.

It allows you to present (**but not demo**) the results of your project in public (speaking opportunity!)

You should use screen shots to describe the finished product

You should summarize the finished product:

its strengths,

its weaknesses,

how well you fulfilled your original plan

a summary of how well the project executed: problems, solutions, successes and failures.

5. General Project Advice.

Tips from Previous Students

- Choose the right project - well defined, not too dependent on third parties, not too ambitious.

- Set up and get everyone to use a common project infrastructure - a mailing list, a version control system, an automatic build and testing process, a bug tracker and regular meetings

- Don't do the work the night before! Work well in advance, and set up (and

respect) individual deadlines when dividing project work

- Choose a good project leader
- Vi använde when2meet för att boka in våra gruppmöten. Detta är något vi kan rekommendera för andra gruppper.
- We noticed early on that we had difficulties gathering solid and valid user requirements. This was primarily due to lack of experience, and secondly due to insufficient communication between the project group and the customer
- Poor requirements led to vague ideas of how to proceed with the project and the documents suffered in quality as a result. It is safe to say that all members of the group now realise how much work goes into a project before actually starting the implementation process.
- The importance of communication within the project group became apparent pretty much when the project started; setting up meetings, distributing tasks and sharing ideas are to name a few essential activities which must be managed. We solved it by using a forum on a website set up and supported by one of the group members, but there are most certainly better forms of communication.
- We encountered on several occasions difficulties in keeping all group members up-to-date on the project, which led to reduced efficiency within the group.
- As tasks amassed — writing documents, code, testing etc. — we divided the group up into smaller groups and distributed the tasks among the subgroups. We found that it generally worked sufficiently, although the number of members within each subgroup may not have been optimal.
- In retrospect, we notice that we lacked a leader: a person who made sure all tasks were completed on time and that the project as a whole moved in the right direction. It would have been comforting if that responsibility had been assigned to only one person and kept that responsibility throughout the project.