Software processes, quality, and standards (course arrangement)

Version 19.11.2018

The latest version is on TUT Moodle (https://ained.ttu.ee/) and on https://tepandi.ee/SQ_org.pdf

TABLE OF CONTENTS

The course arrangement in 2018 differs significantly from the arrangement in 2017. Please use the latest version. The text below will evolve and change during the course.

1 ESSENTIALS

1.1 Basic concepts

The objective of the course is to provide understanding of software quality, its components, different roles in software development, as well as typical processes, methods, skills, and activities related to these roles.

The course include sections on software quality, procurement, development, verification and validation, and maintenance.

It may be useful for an IT manager, procurer, programmer, tester, maintainer, and other stakeholders involved in various software related processes.

Materials

The main materials for the courses are on the TUT Moodle learning environment https://ained.ttu.ee/?lang=en, courses "IDY0204 – Software Quality and Standards" and "ITB8826 – Tarkvara protsessid ja kvaliteet".

In addition, https://tepandi.ee/spqs/ provides basic information for students not having Moodle access.

An example list of materials is provided in Section 6 below.

Prerequisites

The prerequisite for this course is familiarity with basic concepts of software engineering and ability to program in at least one programming language. A prerequisite for passing the examination is working during the semester.

1.2 Evaluation criteria and deadlines

The grading is based on grade points. The activities, deadlines and grade points are as follows.

Activity	Deadline	Max points	Remarks
Lab 1, 2, 3, 4*	Week 5, 9, 12, 14, respectively**	Max 8 points per each Lab***	Uploading Lab 1, 2, 3, 4 to dedicated Moodle forum post. Please see Sections 2 and 7. To receive grade points for a Lab, all previous Labs need to be uploaded. For deadlines please look at the **Note below the table.
Reviews made for Labs 1,2,3,4	Week 8, 12, 14, 16, respectively for each Lab	Max 1 point total per each Lab	A good review gives max 1 point to the author of the review (so, reviews made are individual), and this is also the maximum total number of credit points for reviews per one Lab. To review a specific Lab, the same Lab of the reviewer has to be uploaded. The Lab under review must have the least number of reviews in the time of reviewing (please see Section 3).
Reviews received for Labs 1,2,3,4	Week 8, 12, 14, 16, respectively for each Lab	Max 1 point total per each Lab	A good review gives max 1 point to all the authors of the reviewed Lab (Please see Section 3) , and this is also the maximum total number of credit points for reviews per one Lab.
Examination	Session	60	Please see Section 5

* Note: About Moodle registration and uploading files please see Section 1.3 below.

** Note: All deadlines are until end of week (Sunday evening). Example: Deadline Week 5 = "Until 7.10.2018 at 23:59.59".

*** Notes: To receive grade points for a Lab, all previous Labs need to be uploaded (for example, to receive grade points for Lab 3, Lab 1 and Lab 2 must be uploaded). Late delivery up to one week after the deadline gives 50% of the points received, after that – 0 points. Example: a good Lab 1 presented during week 6 gives 4 points, after week 6 – 0 points.

All Labs must be done and submitted before starting the examination. To pass the examination, min 31 grade points should be received from the examination, min 15 - from Labs.

Grading: 51...60 points = grade "1", 61...70 = grade "2",71...80 = grade "3",81...90 = grade "4", 91 and more = grade "5" ("5" is maximum).

Please do not keep the Labs, reviews, presentations etc to the last date/week, there may be problems with workloads, computers, Internet, seminar time slots etc.

In addition to grade points shown above, additional grade points may be assigned for participation (especially in guest lectures), quizzes, presentations, etc. These points and their arrangement may be announced beforehand, please follow the course Moodle.

General TTU rules apply, such as avoiding violation of academic practices[1].

1.3 Moodle registration, file uploading, other communication

Labs and reviews can only be taken into account for the authors who have registered in Moodle with their full family name + first given name, taken from the student lists in ÕIS. Example: a student "Jack Entoro Dinoder Brass Evan Finion" who has the ÕIS family name "Finion" and given names "Dinoder Entoro Brass Jack Evan" (in this order in ÕIS) needs to register in Moodle with name "Finion Dinoder", otherwise it is very difficult to identify her contributions. Photos in Moodle are welcome.

NB! It is not possible to identify contributions of authors who identify themselves by nicknames in Moodle, so please do not use nicknames.

To enable identification of authors, Labs, and files, please indicate in file uploading the name of the first author, the course code, type of the file (Lab1, Review of Lab1, Lab 2 etc), and topic. Please modify the fields, especially the author's name. Post subject examples:

- "Finion Dinoder, IDY0204. Lab 1. Sports Tracker software for Seawolf Spa".
- "Evelin Smart, IDY0204. Review of Lab 1, Sports Tracker software for Seawolf Spa". [Review for Finion Dinoder Lab 1]

Please indicate the same information in the filenames of all files sent. In Lab files, topics may be omitted. In reviews, topics should be replaced with Lab author name. Example filenames:

- "Finion_Dinoder_IDY0204_Lab1.doc"
- "Finion_Dinoder_IDY0204_ Lab3_tests.zip"
- "Evelin_Smart_IDY0204_Lab1Review_ Finion_Dinoder.doc". [Review for Finion Dinoder Lab 1]

Example student code: 092543IABM.

Example group ID: IVSM11.

Please check the content (next section, about evaluation - also Section 3) and the title list (last section).

In sending other materials, similar principles apply - please enable identification of persons and topics in a large array of e-mails and files, identifying the author, the group, the course identifier, and topic in the Subject line. An e-mail from "Tom <tom112@hotmail.com>" with subject "question" may be

[1] https://www.ttu.ee/faculties/school-of-information-technologies/it-studies/organization-of-studies-2/procedure-for-processing-contemptible-behavior/

lost or indicated as spam by the spam filter. Please send e-mails to your declared lecturers using addresses indicated in Moodle. Example: jaak(dot)tepandi(at)ttu.ee**.**

2 Labs – Acquisition, Development, Testing and verification, Maintenance

The Labs represent different roles in software development and a selection of software life cycle activities associated with each role.

There are four Labs developed by student **teams**. Each team selects its **organisation** and **system** in Lab 1.

Activities in the Labs are practically relevant. Due to study restrictions, they represent a selection and are not covered to full extent. Still where possible, the work done and its description should mirror realistic activities.

The activites grouped into one Lab evolve around the main role for the Lab, but they may be weekly interconnected with respect to their content and data.

The Labs are developed, presented, and evaluated separately, but involve the same team, organisation, and system (with possible exception of code coverage exercise in Lab 2). For convenience they may be below collectively referred to as “project”.

Team. The default project team size is 2, 3, or 4 persons. In case of one person, this must be agreed in writing (eg by e-mail) with the declared lecturer. If all the students have already been included in teams and less than four students remain, then this will be resolved on a case-by-case basis. Responsibility for the project is with the whole team, but each team member should be assigned at least one primary role and associated responsibility.

One team member is the contact person who posts the Labs to the forum and is the first author indicated in the Lab title lists - this is also the contact for correspondence if needed.

All team members must contribute. Adding non-contributing members to the team may lower the amount of credit points for all Labs.

Organisation and system. In Lab 1, each team selects its organisation and system. If an organisation is not available, it can be modelled, imagining as realistically as possible an organisation which needs a system and develops requirements to this system.

The systems / programs used in the project must exist and activities (eg, testing) must be real. The project comprises acquisition, implementation, verification, maintenance activities, but involves only minimal amount of development. Typically it is not possible to develop complex software during the course, so software to be tested in the third Lab should exist: free software, demo version, prototype, functioning application, a program developed in other courses, etc.

The same organization and system is considered throughout the project, with the possible exception in Lab 2.

The project should be as realistic as possible. This means, for example, that both the organisation and the system should be critical and nontrivial, otherwise testing and other activities undertaken are not justified. A good check is "what happens in case of the system failure?" - if the answer includes significant business downtime, financial losses, environmental pollution, injuries etc, then this may be a realistic project topic. Another check is "would the organisation in real life agree to spend significant amount of money and time for tests and other activities done in the Labs?".

Where viable the reports should be presented as a real life documentation, not as a student homework (for example, the task descriptions and questions from the current material should be excluded). One exception from this rule is the Lab title page given in the last section.

The Labs may be related to the projects that are simultaneously carried out by the authors in their organisations. These projects must be ongoing in parallel to the study, copying the request for proposal or other materials from past projects is not acceptable.

The Labs are reviewed by the other students and the lecturers, so there should be no confidential content. Usually this is possible by replacing, removing, obfuscating, or otherwise changing the confidential components. If this is not sufficient, another topic should be taken.

Reports. The report for each Lab should follow the structure given in the below sections. Numerical values associated with activities (eg number of requirements or tests) in the following sections do not depend on the number of persons in the team.

The Lab components in the below sections have a general numbering of activities over all Labs (eg, Lab 2 starts from activity 4, etc), to enable referencing to the sections across all Labs. It is possible to use this numbering in the Labs, or start each Lab from section 1.

Automated test scripts may be added as separate files or appendices; the Labs should be readable without them. If some results are presented as references to components outside the TUT Moodle, these must be publicly accessible for reviewing without registration.

In a realistic system there may be hundreds of requirements and thousands of tests. It is not possible to present them all during a course. So we select just a pre-defined number of artefacts (requirements, tests etc) for a Lab. To have an understanding of the real number of artefacts, some Labs include evaluations of the real number of requirements, tests etc.

For Lab evaluation, please see Section 3 below.

The concepts used in the Labs are explained in the course materials.

Note: the Labs from previous years are not made available, because the Labs for this year differ significantly from the earlier ones. In addition, all Labs have some problems and so these problems would be carried over to the current Labs. Examples of previous Labs are given during the lectures, discussing advantages and disadvantages of each example.

2.1 Lab 1. Acquisition activities (excerpt)

Lab 1 includes selected components of the work description for a request for proposal issued by the organisation (described in activity 1.1) to procure the system (described in activity 1.2).

Main responsible role: Procurer (acquirer, customer, beneficiary).

Activities:

1. **Describing the project.** Components (shortly):

   1.1. Presenting the organisation (and the system to be acquired). The presentation should include at least the following components: organisation and the system to be procured; goals and value of the system for the organisation; system properties that are critical for the organisation (with justifications); stakeholders and their expectations; restrictions on the cost. **Note 1.** Both the organisation and the system must be important (a "homework" would not justify testing etc) and assume non-functional requirements. A control question: would it make sense for this organisation to pay for the activities done in the Labs? **Note 2.** In case there is no real organization available, it can be modelled (this should be realistic, see above - please select an organization which you are familiar with).

   1.2. Introducing the system to be acquired (and organisation). The presentation should include at least the following components: users and usage of the system; expected structure or high-level architecture; components to be procured (for example, code, documentation, etc); rights to be procured. Is this a system, software, service, combination of these? **Note 1.** The system as a whole should be (1) important for the organisation and (2) identifiable (not a module which is not visible to the user). **Note 2**. Description of the system does not need to be extensive – it should just enable the reader to understand the high-level structure of the system (components/applications and understanding of the interfaces between these). Examples: distributed or local? How many layers? How many locations? Communication protocols? Underlying software platforms? Etc. **Note 3.** The system and its testing must be real - there are many systems freely available for download;

or freely available for some period sufficient for testing once the preparations have been made. **Note 4.** Depending on the situation, the system itself may be already selected (in this case this is a call for implementation proposals, and a short explanation is needed why this particular system was selected), or not known (in this case the call is for providing any system which will satisfy requirements, and justification for selection of the system is provided in point 6.5 below).

2. **Developing the requirements.** Requirements must be testable, realistic, and traceable to the customer needs. Requirements may be presented as use cases or executable specifications in Gherkin or similar language (please see Appendix); choice of the presentaion should be justified in the Lab. Each use case includes at least its ID (to be associated with tests), the use case name (represents its goal), actors, preconditions, postconditions, primary scenario. Their choice should be based on quality attributes from ISO / IEC 25010 (other choices have to be justified). Components:

   2.1. **Functional requirements.** At least 10 functional requirements are presented. An evaluation of the realistic number of requirements for this system is given, together with an assessment of how much of the system is covered. **Note.** Evaluation of the realistic number of requirements does not need to be detailed and may be based, for example, on suggesting the approximate overall number of functions and evaluating an average number of requirements per function.

   2.2. **Non-functional requirements.** At least 10 non-functional requirements are presented, including at least two requirements specifying system load properties. An evaluation of the realistic number of requirements for this system is given, together with an assessment of how much of the system is covered.

3. **Planning the acquisition activities - excerpt.** Components:

   3.1. Describe shortly the software development life cycle used. Acquisition activities will depend on it.

   3.2. Participation of the procurer. How much time should procurer's employees plan for this procurement? What equipment, infrastructure, other components will be needed?

   3.3. Change management for requirements. Who and how proposes, discusses, agrees, decides, manages changes? What happens with the schedule and costs?

   3.4. Acceptance plan. How is the system accepted? In which organization (eg customer or supplier)? What is the time schedule? What infrastructure is needed? Who performs the tests? Under which criteria is the system accepted? **Note 1.** The time schedule evaluation does not need to be detailed and may be based on the amount of testing needed (number of requirements), the resources available, and the organisation deadlines. **Note 2.** Acceptance criteria may specify priorities for the acceptance tests (example: certain tests must execute correctly, for certain tests corrective actions are allowed after acceptance), as well as criteria for all properties of the software product and the contractual provisions (examples: installation requirements, data transfer, documentation, user training, licences etc). **Note.** A control question: could a development company make a bid (including amount of work, deadlines, and cost) based on information in Lab 1?

2.2 Lab 2. Development activities (excerpt)

Lab 2 includes exercises for some development activities related to software quality. It also comprises software installation for further testing. **Note 1.** Please see scope definitions for each activity below. **Note 2.** Depending on the selected life cycle, sequencing of activities may differ in Lab 2.

Main responsible role: Developer.

Activities:

4. **Test-driven development - first steps.** Demonstrate first steps of test-driven development (TDD). The programming language used in this exercise may be different from the implementation language of the system specified in Lab 1. System scope covered: a requirement or some requirements from Lab 1. Components:

   4.1. Requirements. For starting the TDD exercise, select some requirements from Lab 1 (if needed, these requirements can be modified to enable implementation in the Labs). In the subsequent TDD cycles, modify or add requirements from Lab 1.

   4.2. Test-driven development cycle. Select a programming language and a unit test tool. Perform at least three TDD cycles (requirement - test - code - refactor; see e.g. Beck, K. Test-Driven Development by Example, Addison Wesley - Vaseem, 2003; or https://en.wikipedia.org/wiki/Test-driven_development)

   4.3. Presentation in the Lab. To enable reviewing, use a version control system publicly accessible for reviewing without registration; manage requirements, tests, and code of each TDD cycle. Document (give references to) the programming language, the unit test tool, and the version control system in the Lab text. Requirements, tests, and code of each TDD cycle can be presented in the Lab text as well. **Note**. This presentation is needed for the Lab review and must enable reproducing the TDD steps done above.

5. **Code coverage.** Demonstrate evaluation of the code coverage. The programming language and the program used in this exercise may be different from the implementation language of the system specified in Lab 1. System scope covered: a program from the system presented in Lab 1, or an independent program. Components:

   5.1. Present the program. Select a program with complexity equivalent of about two pages + 8 if-statements + 4 loops (possibly including interconnected methods / components; if-statements or equivalents, eg replacing conditionals with polymorphism). **Note 1.** It is recommended to select a program from the system to be tested. In case this is not possible, please select an independent program (e.g., searching from the Internet), and present briefly the requirements for this program (or, it is possible to refer to requirements stated in Lab 1). **Note 2.** The idea behind the above complexity requirement is to have branches/iterations to cover by white box testing. Branches are often written as conditional statements, but there are other options. In this case covering all components may be replaced by covering all subclasses, functions, statement components, etc.

   5.2. Coverage tests. Justify and select a coverage criterion for testing. Design tests. Analyse code coverage manually: which program components are covered by the tests, which are not, and why.

   5.3. Test, evaluate. Execute the tests using a unit test tool, evaluate code coverage by a coverage tool. Evaluate the quality of the program and adequacy of testing.

   5.4. Presentation. Present results of previous components in the Lab text or as references to sources publicly accessible for reviewing without registration. **Note**. This presentation is needed for the Lab review and must enable reproducing the code coverage evaluation done above.

6. **Software for further testing.** Install the software for further testing and present architecture (second iteration). System scope covered: the system presented in Lab 1. Components:

   6.1. Installation of the software for further testing. Typically it is not possible to develop full-scale software during the course, so software should exist: free software, demo version, prototype, functioning application, a program developed in other courses, etc. The systems / programs used in the Labs must exist and activities (eg, installation, testing) must be real. At least the following information should be provided for this section:

      6.1.1. Software name, version, author or producer, reference, licence used, duration (if needed).

6.1.2. Installation source reference.

6.1.3. Additional components or activities needed for installation.

6.1.4. Location where the software has been installed, preliminaries required for access.

6.1.5. In case the call was for providing any system which will satisfy requirements (please see Note 4 to point 1.2 above), a short explanation is needed why this particular system was selected.

6.2. Architecture (second iteration). Based on the previous activities, present an additional or enhanced view of the software architecture. This is a second iteration of the architecture, the first iteration of expected structure or high-level architecture was provided in activity 1.2.

2.3 Lab 3.Testing and verification activities (excerpt)

Lab 3 includes selected testing and evaluation exercises with regard to the system, organisation, and requirements presented and installed in the previous Labs.

Main responsible role: Tester.

Activities:

7. Functional specification based testing. System scope covered: the system presented in Lab 1, or its sub-component. Components:

7.1. The component to be functionally tested and detailed requirements. If necessary, the sub-component of the system to be functionally tested is presented and detailed requirements used in functional testing are added.

7.2. Designing functional tests. At least 20 functional tests for the selected sub-component are designed. For each test, at least the requirement(s) used for testing, the equivalence classes and/or boundary situations stemming from the requirement(s), test data, grouping the data into tests, test inputs, and test outputs are presented. The description level should allow for following the test design logic, starting from the requirement and finishing with the test. **Note.** In case these functional tests do not cover all equivalence classes for the selected sub-component, it is necessary to explain the situation and give a short characterization of coverage achieved.

7.3. Saving and running the tests using a test automation tool. Tests are saved and run using suitable tools, for example Canoo WebTest, Selenium, Behavior Driven Development tools (especially if Gherkin or similar language was used for presenting the requirements in Lab 1 or in the steps above), etc. To evaluate and document the results, the tool output (in case it gives readable results), or *ANSI/IEE Std 829 Test-Incident Report* may be used. The test scripts and/or results are added as appendices to the report or as separate files publicly accessible for reviewing without registration. **Note.** If for some reasons automated testing of the selected component is not possible, the tests may be performed manually. In this case additional 20 functional test cases for a freely selected application and suitable tool should be added to the project.

8. Risk assessment and designing acceptance tests. System scope covered: the system presented in Lab 1. Components:

8.1. Risk assessment. Main risks are presented and prioritized based on their potential impact and frequency, using a selected risk ranking method. Each risk is given an identifier that enables to refer to that risk.

8.2. Risk based acceptance tests. At least 20 risk based acceptance tests are designed, addressing the risks in order from highest priority to lowest. The tests must include at least 12 tests for functional requirements, at least 8 tests for non-functional requirements, at least two load tests. The goal is to evaluate whether the system satisfies the most critical requirements. The tests should be inferred from the requirements and risk assessment. Each test should refer to the corresponding requirement (by the requirement ID) and to the corresponding risk (by the risk ID). Tests should cover the requirements as much as possible and be sufficiently detailed so that it is possible to execute them based on the test descriptions. Every test should include at least the test identifier, reference to the requirement (identifier), reference to the risk (identifier), specific given inputs, and specific expected outputs. For documentation also the ANSI/IEEE Std 829 may be used. **Note.** Acceptance tests should not coincide with the functional tests (in a realistic system the number of tests far exceeds the number of tests given in the Labs therefore it is possible to select these tests in a different way).

8.3. Acceptance criteria. Acceptance criteria must be defined. It is possible to refer to activity 3 or to refine the criteria.

9. Preliminary evaluation based on acceptance tests. System scope covered: the system presented in Lab 1. Components:

9.1. Functional acceptance testing. Functional acceptance tests are executed, used for preliminary evaluation, and their results included in the report. Tests can also be saved and executed using appropriate tools, such as the ones used in section 7.3 above. In this case, the test scripts and/or results are added as appendices to the report or as separate files publicly accessible for reviewing without registration.

9.2. Non-functional acceptance testing. Non-functional acceptance tests designed above are saved and executed using appropriate tools, such as Apache JMeter or other. The test scripts and/or results are added as appendices to the report or as separate files publicly accessible for reviewing without registration. **Note 1.** In case the system under test is not web based, other testing tools may be used. **Note 2**. In case the load testing requirements are high and it is not known how the software or site owner will react to high loads, the load testing may be performed with lower actual load, explaining the reason and results in the documentation. **Note 3.** In case some non-functional tests cannot be performed in the ways given above (eg due to some facilities of non-functional testing not available to the testers), it is possible to give an explanation of the situation and present an analysis of the expected results (including the conclusion of whether the author expects the test to pass or not). In this case, additional two test cases using a load testing tool for a freely selected web application must be added to the report.

9.3. Summarising the results of acceptance testing. Based on the above testing and the acceptance criteria, a preliminary summary is given about suitability of the system, covering: variances (report any variances of the test cases from their specifications); comprehensiveness assessment (evaluate the comprehensiveness of the testing process and features that were not suffciently tested); summary of results (summarize the results of testing, identify all resolved incidents and summarize their resolutions, identify all unresolved incidents).

9.4. System evaluation, risk analysis, acceptance. The decision for the evaluated system, eg: accept, develop further, reject, start from the beginning, postpone etc. This evaluation should be based upon the test results and the acceptance criteria defined in Lab 1. What are the risks? Give justifications for the decision.

2.4 Lab 4. Maintenance activities and retrospective (excerpt)

Lab 4 includes selected maintenance activities for the system (scope: as presented in section 1.2) in the organisation (scope: as presented in section 1.1), as well as retrospective analysis, team, and references with respect to the four Labs.

Main responsible role: Maintainer.

Activities:

10. **Proposing system related metrics and forecast.**

    10.1. **Metrics**. Introduce and justify selection of the basic metrics to be used by the maintainer to monitor the system performance.

    10.2. **Forecast.** Propose a significant change in the system due to the tests performed, a request from the customer, or other reasons. Forecast its effort (for example, in man-months) or cost. Describe the method used for the forecast and explain how the method was applied.

11. **Implementing ISKE, ITIL, ISO 9001, or other framework (intro).**

    11.1. **Framework selection.** A framework usable in maintenance, such as ISKE, ITIL, ISO 9001, ISO/IEC 12207, CMMI, or other, is selected for implementation. The selection should be justified based on the project description. In case of other framework its materials should be publicly available (please provide links or content) or presented in the project. **Remark.** A situation where maintenance framework is not needed at all may indicate that the criticality of the project is low.

    11.2. **Overview of implementation.** An overview of the framework implementation. Selection and justification of the necessary processes/activities/components.

12. Retrospective analysis, references, authors. Components:

   12.1. Retrospective analysis. Benefits, lessons, drawbacks form the Labs. Things to be done differently in the future projects – why, how?

   12.2. Presenting the team. The division of work in the team is presented here (please see the team description above).

   12.3. References should cover references to organisation and system (if possible), methods used, etc**.**

2.5 [Optional] Non-standard projects

A non-standard project deals with topics relevant to the course. It should be agreed beforehand by e-mail with the supervisor. Its structure may often be similar to that of the standard Labs presented above (one has to identify stakeholders and the project, analyse requirements, etc).

In case a non-standard project does not involve some practical components of the standard Labs (such as development of white or black box tests, using test tools etc), experience in these should be demonstrated separately.

3 REVIEW

Each student may review another student's Labs from the same semester. This gives feedback to the author and enables improving the work (in case of a timely review). The reviewer has a chance to see virtues and problems of another Lab.

To review a specific Lab, the same Lab of the reviewer has to be uploaded. The Lab selected for the review must have the smallest number of reviews among Labs with the same number (the idea is to exclude situations where one Lab has multiple reviews and another Lab none). The Lab under review must include all parts required at the time of reviewing.

A review has one author (reviews and Lab teams are not related).

A review should be posted in the forum as an answer to the Lab post.

A recommended structure of a review is as follows.

1. **The reviewer data (Author, e-mail, group ID)**
2. **Data of the Lab under review**

- Author(s), group ID
- Lab title, short overview
- Lab status and date
- Review inputs (eg Lab, program, documentation, access to the system etc)

3. Evaluation, taking into account requirements and the structure of the Labs presented above

The reviews should be based on the requirements for Labs from Ch 2. Here are some examples of review questions for different Labs (please check if they are applicable to the specific Lab under review and add questions based on Ch 2):

- Is all the content required in the current version of the course arrangement material present?
- Are both the organisation and the system critical and nontrivial, so that testing and other activities undertaken are justified?
- Are the requirements testable, realistic, traceable?
- Could a development company make a bid (including amount of work, deadlines, and cost) based on information in Lab 1?
- Are risks identifiable? Do risk based tests refer to specific risks and requirements (eg by identifier)?
- Is design of functional tests presented in the Lab (eg equivalence classes and/or boundary situations, test data, grouping the data into tests)?
- Is the program selected for white box testing appropriate? Is test coverage justified by analyzing the program and demonstrating that the coverage has been achieved?
- In case some tests were not possible to perform, are the replacement tests explained and presented as all other tests?
- For all Labs: Is division of work between the team members presented and adequate (each team member should be assigned at least one primary role and associated responsibility; four roles must be assigned for each team - acquirer, developer, tester, maintainer; one team member is the contact person)?
- For all Labs: Did all team members contribute to the Lab according to their assignments (as much as the reviewer is aware)?
- Is documentation correct (eg title list, table of contents, heading numbering?

4. Recommendation for grade points

- Each Lab activity gives max 2,5 points. Formatting and presentation gives additionally max 0,5 points.
- Total recommendation for Lab grade points is given, maximum 8 points.

4 PRESENTATION

Presentation materials and schedule

Presentations are given during regular seminars. The topic should be agreed beforehand by e-mail.

Presentation materials (eg slides) should be sent to the teacher participating in the seminar at least 24 hours before the presentation. The presentations are scheduled according to arrival of the materials (not according to agreements made).

There can be 2...3 presentations during one seminar (about 5...6 in case of a seminar fully devoted to presentations). In case of insufficient time, the first presentation of each author has a priority. In case of a large number of presentations, special sessions can be agreed beforehand.

Authors and duration

A presentation has one author. The recommended duration of a presentation is 10…20 min + discussion. Mutually related presentations may be given in succession, still they are evaluated individually for each author.

Topics

Presentation can be on topics related to the course, such as practical issues of using the V&V methods, tools, and frameworks.

It is assumed that the author and listeners have participated in the classroom sessions, so repetition of the course material does not give additional value and is not suggested for presentation.

A well known topic might be suitable in case the author is an expert on this topic and can present personal practical experience (pluses-minuses-problems etc).

Presentation can also be on interesting quality aspect(s) of a specific software application. The software should be neither too widespread (everybody knows the problems) nor too exotic (nobody is interested). It should rely both on author's experience and overview of references. There should be multiple references discussing various viewpoints (only manufacturer's references are not sufficient). The references should be given in the presentation.

Please include background, references, overview, and example(s) that help to understand the topic and link it to other issues considered in the course. If the presentation is about a tool, please provide a demo. Copy-paste from external sources is not accepted for presentation.

Evaluation

The evaluation of the presentation depends on the timing (presentation made during weeks 1-8 – max 6 GP; during weeks 9-12 - max 4 GP; during weeks 13-16 - max 2 GP; presentation after week 16 - 0 GP); quality of the presentation (content, talk, materials) and the number of listeners (5+ listeners besides the teacher – 100%, 2...4 listeners - 60%; 1 listener – 30%).

Recommendations

Recommendations for the presentation:

- Present yourself and give the author's name and group ID – also on the slides
- Create structure (eg content slide)
- Give some background and overview
- Include different viewpoints
- Avoid copy-paste
- Give sources and references
- Give examples, use demos for tools
- Link the presentation to other issues considered in the course
- Look at and speak to the listeners
- Speak loudly, clearly, and not too fast so that people in back seats can hear
- Watch time

5 EXAMINATION AND QUESTIONS

Arrangement and materials

Examination is oral or written. In oral examination, 6 short questions or problems may be asked (no tickets or materials). Each answer gives max 10 points (total max 60 points, see Section 2). In case of a written examination it may be arranged using multiple choice questions, textual questions, or other methods.

A student can take an examination twice. The last result shall apply.

Selected topics are given in the course slides on TUT Moodle or on tepandi.ee. In addition to the slides, independent reading on these topics is required. Example lists of materials are provided in the course slides and in Section 6 of this file.

Different viewpoints are accepted, but the concepts given in the course should be known, presented or demonstrated during the examination, and compared to other approaches.

Short questions for the examination

- All team members must be able to explain and justify the content of the Labs presented by the team. Explanations and rationale for each Lab component presented may be an examination question.
- Each slide in the course presentations can be viewed as a source for an examination question. Example: "how to develop testable requirements?" (based on the course slides)
- For each method (standard, framework, etc) discussed in the course, its features may be a topic for a question, examples: motivation of the method, idea, preconditions, advantages, disadvantages, results, relationship to other methods, evaluation, tools. An example: "what are the disadvantages of equivalence partitioning?".
- Another source of questions for each topic are the questions of type “why, what, how, who, when, where”. An example: "who typically performs white box testing?".
- The key concepts presented below can be viewed as a source for an examination questions.

Key concepts

The following list presents some key concepts of the course. Please check that they are known among other topics given in lectures.

- Quality, its components. Three quality concepts. Quality as relative. Quality as a trade-off. Four quality models.
- Software as a product. Software and system. Target and non-target software, data, hardware, communications. Layers of complexity / interoperability
- Requirements as intrinsic component of quality. Sources of requirements. From quality model to quality requirements.
- Functional / non-functional + Testable / non-testable + Realistic / non-realistic + Traceable / non-traceable requirements. Risks
- Quality characteristics and subcharacteristics (examples). ISO/IEC 25000 series. Product quality, quality in use, data quality models.
- Software process framework examples. Software development life cycle model examples. Relationship between development life cycle models and process frameworks. Acquisition process. Service level agreements.
- Acquisition, implementation, verification, maintenance processes
- Testing. Test. Verification. Validation. Certification. Debugging. Error, fault, failure.

- Goals of testing? Successful test? Testing in widespread SDLCs.
- Ad hoc, smoke, experience based, exploratory testing
- Risk based testing. Risk parameters. Finding risks. 4T for risk control.
- Price effectiveness of testing methods
- How much to test?
- Difference between systematic and non-systematic testing
- Test automation - idea, possibilities, tools, when to use, recommendations, advantages and disadvantages.
- Specification based (“black box”) testing. Equivalence partitioning and boundary values
- Testing based on the program text (“white box”). Coverage criteria, their comparison
- Non-functional testing. Usability testing. Random testing. Fault seeding
- Testing volume and termination. Price effectiveness of testing methods
- ANSI/IEE Std 829 Standard for Software Test Documentation
- Static methods. Walkthrough/Inspection/Review. Questionnaires. Programmer`s evaluation. IEEE Std 1028 - IEEE Standard for Software Reviews and Audits
- Need for testing management. System and software. Volume of testing needed. Scrum and testing. XP and testing. Test driven development (TDD). Testing with V-model. RUP and testing. Test Process Improvement (TPI). IEEE Std 829 (documentation of tests). Testing automation. Types, advantages, disadvantages of testing tools
- Requirements for critical software. N-version programming. Fault tree analysis. Formal verification. Cleanroom software engineering. Common Criteria for Information Technology Security Evaluation
- Quality management: concepts. Informal QM techniques. Quality awards. ISO 9000 series. Software development, testing and quality management. Quality management – first steps in an organisation
- Agile methods. ITIL / ISO/IEC 20000 series. ISO/IEC 27000 series. ISO/IEC 12207. COBIT. CMMI. Agile and ISO/IEC 12207, CMMI, COBIT
- IT standards and standardization bodies. ISO, IEC, ISO / IEC JTC1
- Software metrics related to product, process, usage, data. Software development effort and cost prediction.
- IT audit. ISACA, COBIT, CISA

6 Additional Reading

Recommended materials for additional reading are provided below.

- References given in the slides of the Moodle course "Software quality (Tarkvara kvaliteet)" or tepandi.ee
- Guide to the Software Engineering Body of Knowledge (SWEBOK), IEEE, http://www.computer.org/portal/web/swebok
- Certified Tester Foundation Level Syllabus. International Software Testing Qualifications Board, http://www.istqb.org/download.htm + Other materials from this site

- Ian Sommerville. Software Engineering. Ninth Edition. Addison-Wesley
- Daniel Galin. Software Quality assurance from theory to implementation. Pearson - Addison-Wesley
- ISO/IEC 25010. Software engineering: Software product Quality Requirements and Evaluation (SQuaRE) — Quality model
- IEEE Std 829. IEEE Standard for Software and System Test Documentation
- COBIT, http://www.isaca.org/cobit
- Perry W. Effective Methods of Software Testing. John Wiley & Sons, Incorporated
- Robert C. Martin. Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall, 2008
- Pressman, R. Software Engineering: a Practitioners Approach
- Books including software quality, testing, and standards issues as significant portions of the text, edition after 2000 preferred. Example: G. O'Reagan, A Practical Approach to Software Quality, Springer
- Curriculum Guidelines in Software Engineering, http://www.acm.org/education/curricula-recommendations
- CMMI - http://cmmiinstitute.com/
- TPI - http://www.tmap.net/en
- General books on software engineering, management and standards as a complementary material

7 Appendixes

7.1 Lab structure, format, and title page

The preferred structure of a Lab follows the structure given in Section 2. The title page (see next page) and the table of contents are needed. The sections and sub-sections must be numbered. Examples of past Labs are given during lectures. As the Labs may vary significantly depending on the organisation and system, files of previous Labs may be misleading and are not given.

The next page presents a Lab title page. The title reflects the content (not "homework" etc). Given names, family names, and group IDs of all authors are presented. Please change the fields as necessary.

To keep the evaluations confidential, please do not indicate the student codes on the title page.

The evaluations and remarks on the Labs will be available on the course website, arranged by the student codes.

The first author indicated in the Lab title list should post the Lab to the forum - this is also the contact for correspondence if needed.

TALLINN UNIVERSITY OF TECHNOLOGY

DEPARTMENT OF SOFTWARE SCIENCE

Accounting Software for the PeopleDesign Ltd
Request for Supply (excerpts)

Lab 1 in subject "Software Quality and standards" (IDY0204)

Authors, group ID [no student codes]
..
..
..
..
Group ID [eg, IVSM] ………………………
Presented: ……………………………………
Supervisor: ....

TALLINN 2018

7.2 Example templates for Labs

This section gives some examples of templates that can be used in the Labs. The templates are **non-mandatory and kept to the minimum**, please adjust them according to your task as needed.

In particular, if the Labs are used in a real-life project, please feel free to use templates provided by your organisation.

7.2.1 Example templates for functional requirements

The table below provides an example use case template for functional requirements.

Use Case ID	
Use Case Name	
Primary Actor	
Preconditions	
Postconditions	
Main Success Scenario	

Where applicable, requirements may be presented as user stories, please see https://en.wikipedia.org/wiki/User_story).

It is also possible to use executable specifications, e.g. in the Gherkin language.

Comparison of use case / user story and Gherkin representations:

- Use case representation is generic, providing a whole function. Gherkin representation may also present a function, but the function itself is not tested - only test cases are tested
- Use case representation is universal (most requirements can be represented as use cases)
- Gherkin representation may be better implementable and testable
- Gherkin representation may be better testable, potentially less ambiguous, etc

7.2.2 Example Use Case template for non-functional requirements

For non-functional requirements, it is possible to use the above templates, to modify them, or to use Service Level Agreements. Example of a modified template:

Use Case ID	
Use Case Name	
Success Criteria	

7.2.3 Example template for designing functional tests

An example template for designing functional tests based on equivalence classes and boundary values can be found in the functional testing practice.

7.2.4 Example templates for risk assessment

Risks can be prioritized based on their potential impact and likelihood, using a selected risk ranking approach. An example of risk ranking is based on the table below (from ISO/IEC 27005), where risk measure is given in the table on a scale from 0 to 8. This risk scale can also be mapped to an overall risk rating, for example: low risk (0-2), medium risk (3-5), high risk (6-8). The table below is symmetrical, but it many cases this is not the case.

		Likelihood of incident				
		Very low / Very unlikely	Low / Unlikely	Medium / Possible	High / Likely	Very High / Frequent
Business impact	Very low	**0**	**1**	**2**	**3**	**4**
	Low	**1**	**2**	**3**	**4**	**5**
	Medium	**2**	**3**	**4**	**5**	**6**
	High	**3**	**4**	**5**	**6**	**7**
	Very high	**4**	**5**	**6**	**7**	**8**

More examples of risk measures are given on lecture slides, on https://en.wikipedia.org/wiki/IT_risk, etc. Please select a method in the Lab according to the particular organisation and system provided in Lab 1.

Each risk is given an identifier that enables to refer to that risk. Risks are prioritized according to their ratings. For example, in the case of table above, the risks with rating 8 would have the highest priority (rank). They need to be tested first. Below is an example table of risk prioritization (which risk needs to be tested first?).

Risk ID	Risk description	Likelihood	Impact	Risk rating	Priority
R1	………….	Low	Medium	3	3
R2	……………	Very High	Medium	6	1
R3	…………….	Very Low	Very High	4	2

7.2.5 Example templates for risk based acceptance tests

For risk based acceptance tests, it is possible to use different formats. If the test inputs and expected outputs are short, the following format can be used, modifying it as needed. The highest priority risks are tested first.

Test ID	Req ID	Risk ID	Idea	Input	Expected output
AFT1	NFR2	R2	……..	………….	…………….
…					
AFT12					
ANFT1					
…					
ANFT8					

To make tests repeatable, additional information may be needed, such as execution preconditions or postconditions etc. For tests that have long input-output descriptions and/or pre- or postconditions, the following format may be modified.

Test ID	AFT1
Requirement ID	
Risk ID	
Idea	
Preconditions	
Input	
Expected output	

7.2.6 Example template for a test incident report

Discrepancies between the actual and expected test outcomes are presented as incidents. A test incident report may include the following.

Test incident report ID	TIR1
Test ID, date, tester	
Summary	
Inputs	
Expected results	
Actual results	
Severity	
Recommendations	
References	