Software processes, quality, and standards (course arrangement)

Version 10.10.2017

The latest version is on Moodle course "Software quality (Tarkvara kvaliteet)" and on tepandi.ee

TABLE OF CONTENTS

The course arrangement in 2017 differs significantly from the arrangement in 2016, please use the latest version.

1 ESSENTIALS

1.1 Basic concepts

Course aims

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.

Learning outcomes

Upon successful completion of this course, a student will be able to:

- understand software quality management, verification and validation, and standardization principles, processes, and methods
- apply software verification, validation, and quality management methods and tools
- understand program testing principles, select and apply testing methods, and evaluate system quality
- choose processes, methods, and activities related to different roles in software development

Brief description of the course

The first part of the course covers the notion of software quality and its intrinsic components - software, requirements, and processes, including the acquisition process. The second part deals with methods and management of software verification and validation: testing (the principles and methods of software testing, white box, functional, other methods), static methods (inspections, reviews and walkthroughs, program correctness proving), and the frameworks for testing management. The third part provides a selection of software quality management frameworks, processes, and standards topics, including the principles of quality management and standardization, software process quality, as well as software quality criteria and metrics.

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

The main concepts are provided on the slides and other materials on the course website - Moodle course "Software quality (Tarkvara kvaliteet)". In addition, participation and extensive independent reading is required.

Various tools are used in the course project. Selection of the tools is not fixed - as the tools are in continuous development, the students are encouraged to find and apply the tools most suitable for their tasks.

Study literature

An example list of materials is provided in Section 6. For an overview, materials from the first references of Section 6 are recommended.

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.

Declarations

Depending on the study group, the courses should be declared and examinations given to the following lecturers:

- IAPM, IABM regular courses - Stanislav Vassiljev
- IABM distance learning (IDX1511) - Jekaterina Tšukrejeva
- IVSM11 ja IVSM12 (IDY0204) - Jaak Tepandi

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, 8, 11, 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, 11, 14, 16, respectively for each Lab	Max 1 point per each Lab	A good review gives max 1 point to the author of the review (so, reviews made are individual). 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, 11, 14, 16, respectively for each Lab	Max 1 point per each Lab	A good review gives max 1 point to all the authors of the reviewed Lab (Please see Section 3).
Participation	Week 16	14	Grade point tests, participation checks, discussions. Grade point tests will be performed during regular lessons and announced beforehand. Participation checks may be or not be announced beforehand.
Presentation***	Week 16	6	One presentation gives max 6 p, and so do all presentations in total. Please see Section 4.
Examination	Session	40	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: "Week 5" = "Until 8.10.2017 at 23:59.59".

*** Note: Late delivery or presentation 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. 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).

Examination and Labs are mandatory. To pass an examination, min 15 grade points should be received from the examination, min 15 - from Labs and other activities.

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.

General TTU rules apply.

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, projects, 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 lost or indicated as spam by the spam filter. Please send e-mails to your declared lecturers using addresses indicated in Moodle**.**

2 LABS – ACQUISITION, IMPLEMENTATION, 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.

The Labs are developed and evaluated separately, but involve the same team, organisation, and system. For this reason they are also collectively referred to as “project”.

Team. The recommended project team size is four persons. In case of more team members, this must be agreed in writing (eg by e-mail) with the declared lecturer. Responsibility for the project is with the whole team, but 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 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. This division of work must be presented in the project.

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 in general does not include coding. Typically it is not possible to develop software during the project, so software 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 for program based tests (see Lab 2). Where possible the report should be presented as a project 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 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.

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 the whole project. 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 project should be readable without them.

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 of the project 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.

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

2.1 Lab 1. Acquisition activities (excerpt)

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

Main responsible role: Acquirer.

Activities:

1. **Describing the project.** Components:
   - 1.1. **Presenting the organisation (and the system to be acquired).** The project 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. **Remark 1.** Both the organisation and the project 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 project? **Remark 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 project should include at least the following components: users and usage of the system (what is the difference between a stakeholder and a user?); expected architecture; components to be procured (for example, code, documentation, etc); rights to be procured. **Remark 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). **Remark 2**. Description of the system does not need to be extensive – it should just enable the reader to understand the high-level architecture of the system (components/applications and the definition of the interfaces between those components and applications). Examples: distributed or local? How many layers? How many locations? Communication protocols? Underlying software platforms? Etc. **Remark 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 project has been prepared. **Remark 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 4.5 below).

2. **Developing the requirements.** Requirements must be testable, realistic, and traceable to the customer needs. Requirements are presented as use cases (also, where applicable, requirements may be presented as user stories, please see https://en.wikipedia.org/wiki/User_story). 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 or ISO/IEC 9126 (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 assessment how much of the system is covered. **Remark.** 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 assessment 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 to this procurement? What equipment, infrastructure, other components will be needed?

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

   3.4. Acceptance plan. How is the system accepted? In which organization (eg customer or supplier)? What is the time schedule? Who performs the tests? Under which criteria is the system accepted? **Remark 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. **Remark 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 - please see the presentation on basic concepts of the course).

2.2 Lab 2. Implementation activities (excerpt)

Lab 2 includes selected implementation activities to install, review, and test the system presented in activity 1.2.

Main responsible role: Developer.

Activities:

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

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

   4.2. Installation source reference.

   4.3. Additional components or activities needed for installation.

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

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

5. **Planning and performing walkthroughs.** Walkthroughs (or other static V&V methods) in the project are planned according to the problem statement. Who will perform them, how often, who will participate, how will the results be used? At least one walkthrough is performed. **The object, participants, activities and results of this walkthrough are presented.** The walkthrough type is selected by the author according to the organisation and the testing task. **Remark.** The system must be sufficiently important and critical to justify performing walkthroughs or other static methods.

6. **White-box testing.** Components:

   6.1. **Present the program.** Select a program (at least two pages + 8 if-statements + 4 iterations) and add its text to the project. **Remark 1.** It is recommended to select a program from the system to be tested (a system may include hundreds of program units). In case this is not possible, please explain the situation, select an independent program (eg searching from the Internet), and present briefly the requirements for this program. **Remark 2.** The selected program may include several interconnected methods / components.

   6.2. **Coverage tests.** Justify and select a coverage criterion for testing, design tests, present as in the previous section. **Remark.** The number of tests is not pre-determined (why?).

   6.3. **Test, evaluate.** Perform the designed tests, document the results, evaluate the quality of the program and adequacy of testing. Test tools may be used, for example JUnit or others. **Remark.** In this section of the project the program selected in part 6.1 is analyzed. All other Labs discuss the initial system described in Lab 1 (or its components).

2.3 Lab 3. Verification activities (excerpt)

Lab 3 includes selected testing and evaluation activities of the system (presented in section 1.2) in the organisation (presented in section 1.1) with respect to requirements and acquisition procedures (presented in activities 2 and 3).

Main responsible role: Tester.

Activities:

7. **Functional specification based testing.** 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. **Functional tests.** At least 20 functional tests for the selected sub-component are designed. For each test, 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, and the tests themselves are presented. The description level should allow for following the test design logic, starting from the requirement and finishing with the test. The tests are presented similarly to acceptance tests. The test scripts and/or results are added as appendices to the report or as separate files. **Remark 1.** 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. **Remark 2.** Functional tests should not coincide with the acceptance tests (in a realistic system the number of tests far exceeds the number of tests given in the project therefore it is possible to select these tests in a different way).

   7.3. **Saving and running the tests using a test automation tool.** Tests are saved and run using appropriate tools. 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.

8. **Risk assessment and designing acceptance tests.** Components:

   8.1. **Risk assessment.** Risks are prioritized based on their potential impact and frequency. 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. The tests must include at least 12 tests for functional requirements, at least 8 tests for non-functional requirements, and 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 the ANSI/IEEE Std 829 may be used.

   8.3. **Acceptance criteria.** Acceptance criteria must be defined (please see activity 3).

9. **Preliminary evaluation.**

   9.1. **Functional acceptance testing.** Functional acceptance tests are saved, used for preliminary evaluation, and included in the project. Tests are saved and executed using appropriate tools, such as Canoo WebTest or Selenium; in case the source code is available, JUnit is also an alternative, but in this case testing must still be on acceptance testing level (not on the level of testing program logic). The test scripts and/or results are added as appendices to the report or as separate files. **Remark 1.** In case the system under test is not web based (or the proposed tools cannot be used for some other reason), other testing tools may be used. **Remark 2.** In case the system under test is not web based and other testing tools cannot be found, the tests may be performed manually. In this case additional 10 test cases for a freely selected web application should be added to the project.

   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. **Remark 1.** In case the system under test is not web based, other testing tools may be used. **Remark 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. **Remark 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 jMeter 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 items 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), evaluation (provide an overall evaluation of each test item including its limitations; this evaluation should be based upon the test results and the item level pass/fail criteria; an estimate of failure risk may be included).

   9.4. **System evaluation, risk analysis, acceptance.** The decision for the evaluated system, eg: accept, develop further, reject, start from the beginning, postpone etc. What are the risks? Give justifications for the decision.

2.4 Lab 4. Maintenance activities and retrospective (excerpt)

Lab 3 includes selected maintenance activities for the system (presented in section 1.2) in the organisation (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 basic metrics to be used by the maintainer to monitor the system performance.
    - 10.2. **Forecast.** Propose a significant change in the system on the basis of the previous experience and tests. Forecast its cost.
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 available 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. **Security classes (in case of ISKE).** In case of ISKE: analysis and justification of security classes.
    - 11.3. **Processes (in case of ITIL).** In case of ITIL: an analysis of which processes should be used.
    - 11.4. **First activities (in case of ISO 9001).** In case of ISO 9001: an analysis of practical activities for starting implementation of ISO 9001 for the selected organisation (and system).
    - 11.5. **Implementation (in case of other framework).** In case of other framework: a short overview of implementation, selection and justification of the necessary processes/activities/components. As an example, this might include use of effect mapping in change management.
12. **Retrospective analysis, references, authors.** Components:
    - **12.1. Project retrospective analysis.** Benefits, lessons, drawbacks form the project. Things to be done differently in the next project – why, how?
    - **12.2. Presenting the team.** The recommended project team size is four persons. In case of more team members, this must be agreed in writing (eg by e-mail) with the declared lecturer. Responsibility for the project is with the whole team, but 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 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. This division of work must be presented here.
    - **12.3. References** should cover the problem statement (eg the current material, probably other sources), methods used, organisation and system (if possible), etc**.**

2.5 [Optional] Non-standard projects

A non-standard project deals with topics relevant to the course and agreed beforehand by e-mail with the supervisor. Its structure may often be similar to that of a standard project (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 presented above (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 project.

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 project 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. **The Lab data**
    - Author(s), group ID
    - Lab title, short overview
    - Lab status and date
    - Review inputs (eg project, 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 non-functional requirements testable?
- Are risks identifiable? Do risk based tests refer to specific risks and requirements (eg by identifier)?
- Is design of functional tests presented in the project (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?
- 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

Authors and duration

A presentation has one author. The duration of a presentation is usually about 15 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.

Presentation materials and schedule

Presentations are given during regular seminars. They should be agreed beforehand by e-mail. Presentation materials (eg slides) should be sent to the teacher participating in the seminar at least one day 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.

Evaluation

The evaluation of the presentation depends on the 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
- Give sources and references
- Give examples, use demos if possible
- 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, 5 short questions or problems may be asked (no tickets or materials). Each answer gives max 8 points (total max 40 points, see Section 2). In case of a written examination it may be arranged using multiple choice questions or other methods.

Selected topics are given in the course slides on Moodle course "Software quality (Tarkvara kvaliteet)" 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

- 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 slides from Jaak_Tepandi_Sw_QS_00.pdf)
- 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. 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 course basics

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. Software development effort and cost prediction. COCOMO
- 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 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 projects are given during lectures. As the projects may vary significantly depending on the organisation and system, files of previous projects 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 project will be available on the course website, arranged by the student codes.

The first author indicated in the Lab title list should post the project 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 2017

7.2 Template examples

This section gives some examples of templates that can be used in the labs. The templates are **non-mandatory**, 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 Use Case template for functional requirements

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

7.2.2 Example Use Case template for non-functional requirements

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

Use Case ID	
Use Case Name	
Success Criteria	