**Week 2 notes**

Cohesiveness:凝聚力

**WEEK 3 Software Quality Plan**

<https://flip.ee.usyd.edu.au/elec5618/_downloads/sqp_notes.pdf>

05\_sqp\_section

**WEEK 4 SRS and use cases**

**WEEK 5 Verification**

SELF-ASSESSMENT: <https://flip.ee.usyd.edu.au/elec5618/Material/Verification_and_validation/Basic_questions/vandv_read_notes.html>

Verification vs validation notes: <https://flip.ee.usyd.edu.au/elec5618/_downloads/vandv_notes.pdf>

**WEEK 6 software testing**

Notes:

<https://flip.ee.usyd.edu.au/elec5618/_downloads/sw_testing_notes.pdf>

**Week 7 The software test plan**

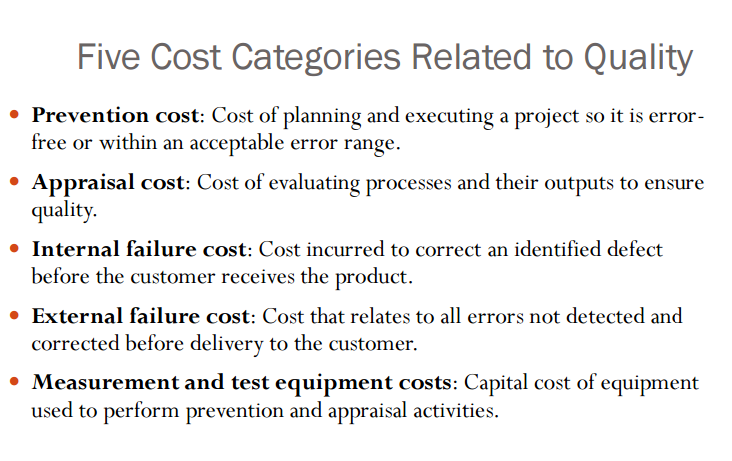
Notes: <https://flip.ee.usyd.edu.au/elec5618/_downloads/test_plan_notes.pdf>

**Week 8 Tools for testing (FSM,MARKOV CHAINS)**

Notes: <https://flip.ee.usyd.edu.au/elec5618/_downloads/other_tools_methods_notes.pdf>

**Week 9 SQE in agile environments**

Notes: <https://flip.ee.usyd.edu.au/elec5618/_downloads/agile_development.pdf>



|  |  |
| --- | --- |
| Terms |  |
| Quality planning | Identifying which quality standards are relevant to the project and how to satisfy them. Select applicable procedures and standards for a particular project and modify these as required.  how we make that level of quality happening. What method should we employ to reach that much quality. |
| Quality assurance | Periodically evaluating overall project performance to ensure the project will satisfy the relevant quality standards.  how we monitor the workflow/environments so that the work shouldn't go astray. |
| Quality control | Monitoring specific project results to ensure that they comply with the relevant quality standards. Ensure that procedures and standards are followed by the software development team.  is kind of feedback of the final product and the product must pass to meet the satisfactory criteria. |
| The people capability maturity model | Intended as a framework for managing the development of people involved in software development.  PCMM has 5 stages that describe level of competency and behavior of human resources, ranging from initial ad-hoc, to advanced personal who can develop himself. It can help get persons move up the level, ie. improving. It's applicable everywhere that has people in it, software development included.   * 提高人力的成熟度以此来提高组织的成熟度 * 软件开发工程不能只依赖与一小部分的人 * 帮助人力保持他们的技术(RETAIN)   有五个阶段   * 第一阶段intitial,只要少数人能够开发软件，一旦人员改变，就会很难。 * 第二阶段repeatable:一些基本的方针制定以应对一些危机状态 * 第三阶段defined * 第四阶段managed：计量目标 * 第五阶段optimizing: 继续帮助个人的能力以及提高人力的积极性 |
| Verification | Are we building the product right |
| Validation | Are we building the right product |
| SRS | It is used to define the following points:   * Functional requirements * Nonfunctional requirements such as reliability, security, performance, usability, availability and portability constraints.    It's used by all stakeholders to have the same understanding of the final product. It's in some way a compromise that everyone will have to accept the product if it performs as described in the requirements. |

WEEK 3 NOTES

|  |  |
| --- | --- |
| Quality Plan(IEEE 730-2002) - documentation | * Software Requirement Specification: A description of what should the system do. * Software Design Description: A description in blocks of the intended implementation to comply with the software requirements. * Verification and validation plans: Describe the processes what will be used to verify and validate the application. In here, tasks such as code analysis, inspections, and testing are included. * Verification and validation results report. * User documentation.   Software configuration management plan. |
| Standards | reliability, security, performance, usability, availability and portability |

**WEEK 5 Verification vs Validation**

* Static and Dynamic V&V
* Program testing and debugging
* V&V Planning: The software test plan
* Inspections and automatic static analysis

|  |  |
| --- | --- |
| Two principles | The discovery of defects in a system  The assessment of whether or not the system is usable in an operational situation. |
| Static verification | Software inspections Concerned with analysis of the static system representation to discover problems |
| Dynamic verification | Software testing Concerned with exercising and observing product behaviour |
| Stages of static analysis | * Control flow analysis. Checks for loops with multiple exit or entry points, finds unreachable code, etc. * Data use analysis. Detects un-initialised variables, variables written twice without an intervening assignment, variables which are declared but never used, etc. * Interface analysis. Checks the consistency of routine and procedure declarations and their use * Information flow analysis. Identifies the dependencies of output variables. Does not detect anomalies itself but highlights information for code inspection or review * Path analysis. Identifies paths through the program and sets out the statements executed in that path. Again, potentially useful in the review process * Both of these last 2 stages generate vast amounts of information. They must be used with care. |
| Differences | Validation refers to make sure a product complies with its requirements. Verification refers to checking that the product functionality executes correctly. |
| Objectives | Detection of defects and assess if the system is ready to be used. |
| Main aspects that affect the level of confidence of them | The user expectations, how critical is the software for the company, and how soon the product must be in the market. |
| Tools for automatic | The tools for automatic software verification are NOT only used for Dynamic Verification |
| Dynamic/Static verification | Software inspections are *static* verifications  software tests are *dynamic* verifications.  Requirements, High-level design, Formal specification, Detailed design, and program (the code) can be all statically verified. |
| Tests on program | If all tests passed, which only means that the tests did not discover any errors and nothing else. |
| Code inspection | We need the code, description of the system, error checklist to do a code inspection |
| Automatic Static Analysis | Errors that can be deduced by parsing the code. |
| Fagan Inspection | 1. Advantages: The number of errors in the final product can significantly decrease, creating a higher quality product. 2. Disadvantages: Some common errors would repeatedly being reviewed. |
| Difference between code inspection and a code walk through | Inspection: Formal, initiated by the project team, planned with fixed roles assigned to all the members involved, reader reads the product, recorder records the defects  Walkthrough: informal, initiated by the author, unplanned, author reads the code, author makes note of defects |

**WEEK 6 Software Testing**

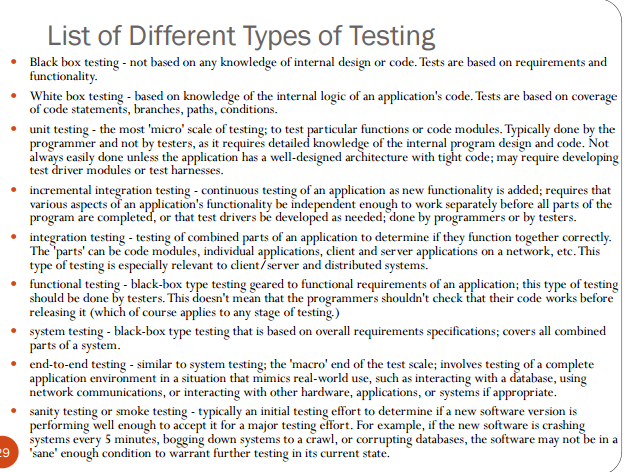
* **Component vs Integration testing.**
* **Structure of a test case: data, result, reports.**
* **Black box testing.**
* **Equivalence Partitioning**
* **Structural testing: program flow graphs.**

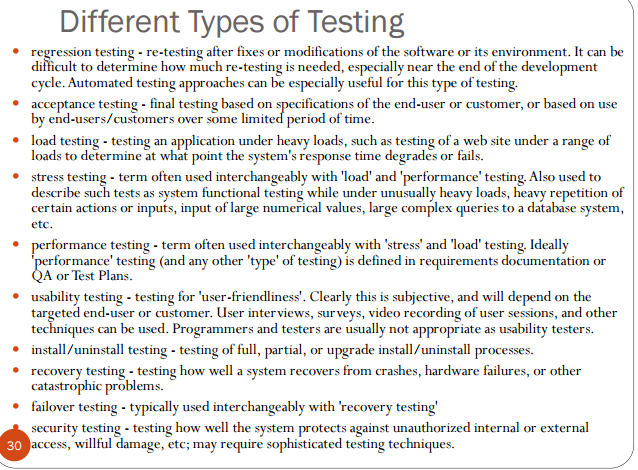
|  |  |
| --- | --- |
| Component testing | Testing of individual program components. The responsibility of the component developer |
| Integration testing | The responsibility of the independent testing team.  Testing of groups of components integrated to create or sub-system. |
| Tests on integration testing | Tests that are performed over the *entire system* are a special case of the so-called *integration testing* |
| Unit testing | Its component testing. |
| Input Equivalence Partitioning | A technique to divide the tests based on categories of the input values. |
| Pre-condition when describing a function | A condition that must be true before the execution of the function for it to produce the correct result. |
| Essential feature of “white box testing” | The tests are derived by exploring the structure of the code. |
| Program flow graph | A function needs to be executed in order to obtain the program flow graph. (FALSE)  Two functions that implement the same functionality, must have identical program flow graphs. (FALSE) |
| Cyclomatic complexity | Know the number of tests so that all control statements are tested once. |
| Comparison on BBT and WBT | Type of code portion:  It is used to test the overall system, it doesn’t take into account the method implementation, it only cares about getting the correct output values based on certain input.  White Box Testing is used to test smaller components and units of the system focusing on validating the internal implementation and structure, as well as output.  When is it used:  WBT is usually on smaller parts of code it would be done earlier on whereas BBT would be done later as assuming code already works so dont need to know internals.  Type defects:  BBT focuses on the output matching the expected output based on certain input.  WBT focuses on internal execution errors, design flaws and algorithm inefficiencies.  Detects defects easier to fix:  The information the BBT gives is that the only output value does not match the expected value, it is difficult to determine where it went wrong which make them hard to fix  WBT can detect simpler defects since it is about certain components and pinpoints exactly where the defect is.  Who is performer:  Testers or programmers can perform BBT since it does not require knowledge of implementation.  Programmers can perform WBT since it requires deeper understanding of the software system. |
| Cyclomatic complexity | = Number of edges - Number of nodes +2 |

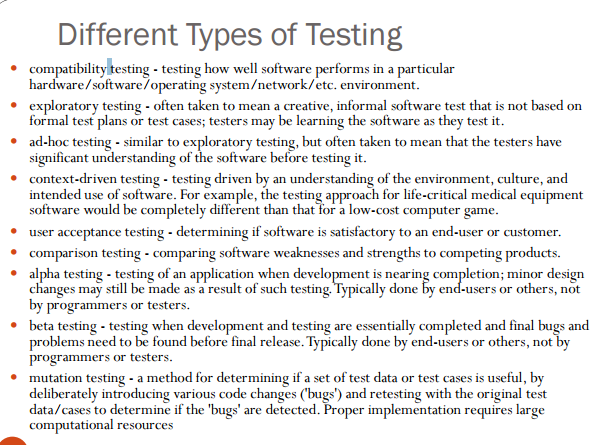
**Week 7 Testing Plan and Types of tests**

* Content of a Test Plan.
* Example of type of tests in the plan: functional, integration, user acceptance, performance, etc.
* Roles and responsibilities of your test team.
* Writing a test case.
* Integration testing.
* Top/down vs Bottom/up integration testing.
* Interface testing.
* Object oriented testing.
* Scenario based testing.
* List of different types of testing.

|  |  |
| --- | --- |
| Functional Testing | Testing to ensure the functional requirements of the program are being met.  E.g. The objective of this test is to ensure that each element of the application meets the functional requirements of the business as outlined in the : Requirements Catalogue |
| Integration testing | Its aim to ensure all components of program are being integrated correctly.  E.g. This test proves that all areas of the system interface with each other correctly and tLEhat there are no gaps in the data flow. Final Integration Test proves that system works as integrated unit when all the fixes are complete. |
| User acceptance testing | Ensuring that the end users business requirements are being met by the system  E.g. This test, which is planned and executed by the Business Representative(s), ensures that the system operates in the manner expected, and any supporting material such as procedures, forms etc. are accurate and suitable for the purpose intended. It is high level testing, ensuring that there are no gaps in functionality. |
| Performance testing | Ensuring the performance of the system is as documented in terms of speed, reliability and other performance measures.  E.g. These tests ensure that the system provides acceptable response times (which should not exceed 4 seconds). |
| Top-down testing | Start with high-level system and integrate from the top-down replacing individual components by stubs where appropriate |
| Bottom-up testing | Integrate individual components in levels until the complete system is created |
| Approaches to integration testing | Top-down integration testing is better at discovering errors in the system architecture  Test implementation: Often easier with bottom-up integration testing  Top-down integration testing allows a limited demonstration at an early stage in the development |
| Interface testing | Takes place when modules or sub-systems are integrated to create larger systems |
| Approaches to cluster testing | Scenario-based testing: Testing is based on a user interactions with the system  Thread testing: Tests the systems response to events as processing threads through the system  Object interaction testing: Tests sequences of object interactions that stop when an object operation does not call on services from another object |







* **Title**: (Title)
* **Type of test**: (unit, integration, black, white, path, scenario, ...)
* **Purpose**: (short description)
* **Prerequisite**: (requirements before the test is executed)
* **Input data/Entry criteria**: (list of vars and values)
* **Steps**: (list of steps to carry out the test)
* **Output**: (expected)
* **Exit criteria**: (describe how to determine the pass/fail outcome.
* **Recommendations**: (open text)
* **Note**: (open)

**Week 8 Tools for testing**

* How to enhance testing strategies.
* Understand the concept of single state mode.
* Justify the use of FSMs.
* Description and use of FSMs.
* FSM Representation.
* FSM creation.
* From FSM to Markov chains.
* Test management tools.

|  |  |
| --- | --- |
| Finite State Machine | Use of FSM offer a more expressive testing mechanism that is more appropriate for certain applications.  输出只和状态有关，就做MOORE  输出和状态和输入都有关系叫做MEALY  输入和当前状态转移到另一个状态叫做状态转移 |
| Steps | Program starts in the “initial” state.  User clicks in “File” menu.  The File menu appears.  User clicks in “Save” item  The File menu disappears.  File is saved  Program goes back to the “initial” state |
| Exercises | <https://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Problem_Solving/Finite_state_machines> |
| Deterministic | Transitions are deterministic: Given a state and an input, a single next state is possible. |
| Markov Chain | Sum of probabilities of all out-going transitions from a state are 1.  Exercise those paths that are more likely to be used.  Transitions in FSMs are extended with a probability value. |
| Testing Management Tools |  |
|  |  |
|  |  |

Testing Management tools 还没看

**Week 9 SQE in Agile Environments**

* Review the content of the Validation and Verification notes with special attention to:
  + CMM: Where is its emphasis
  + Agile: How is it defined
  + Agile: How is it different
* Review the propositions on how to adopt a Scrum type of management scheme in an Agile development environment.

A description of what is the Scrum management framework and how can it be used within an agile development environment.

|  |  |
| --- | --- |
| Scrum management framework | Scrum is an agile software development model based on multiple small teams working in an intensive and interdependent manner. It is a management and control process that cuts through complexity to focus on building software that meets business needs. Scrum employs real-time decision-making processes based on actual events and information. This requires well-trained and specialised teams capable of self-management, communication and decision-making. Thus Scrum itself is a simple framework for effective team collaboration on complex software projects. |
| How Scrum used in an agile development environment | Enumerate the three most significant guidelines that should be followed to adopt this management strategy in an agile development process.  1) Develop small, incremental releases and iterate  2) Complete each feature before moving on to the next  3) A collaborative & cooperative approach between all stakeholders (team members) is essential |
| CMM versus Agile | CMM:  Advantages: CMM driven process improvement also delivers real cost savings such as earlier and more effective error detection  Disadvantages: CMM certification focuses more on an organisation’s management processes and less on the quality of the software products produced  Agile:  Disadvantages: There is lack of emphasis on necessary designing and documentation.  In case of some software deliverables, especially the large ones, it is difficult to assess the effort required at the beginning of the software development life cycle. |
| Agile (advantages) | * Satisfy the Customer through early and continuous delivery of valuable software. * Business people and developers work together daily throughout the project. * Face-to-face conversation is the most efficient and effective method of conveying information * Constant feedback – stay on track through daily short meetings * Customer involvement using facilitated workshops or focus groups |
| Pair Programming | All programming is done by pairs of programmers working so closely together that one can pick up and/or modify the work of the other at any time. |
|  |  |
|  |  |
|  |  |