Chapter 6: Testing

* Verification: determining whether workflow has been carried out. Validation: intensive evaluation process before product is delivered.
* Fault is in the software, human makes a mistake, causes a failure in the software product, error is amount by which it’s incorrect.
* Non-execution-based testing = testing software without running any test cases
* Walkthrough teams/SQA (software quality assurance) teams inspect.
* Inspection

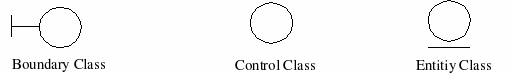
1. Overview of the document is inspected, and then distributed to participants.
2. Preparation is the participants understanding the document in detail. Helps orient teams to where faults might happen.
3. One participant goes through code with inspection team, ensuring each item is covered. Then faultfinding commences. Purpose is to find and document faults, not correct them. Written report ensues.
4. Rework is resolving faults.
5. Follow-up is the moderator ensuring that every issue raised has been solved. If more than 5% has been reworked, team must re-inspect.

* Reader leads the team through design, recorder responsible for producing written report of detected faults.
* A review: effective way to detect a fault, especially earlier on. Inspection rate are when specifications and designs are inspected, the number of pages inspected per hour can be measured. Fault density, measure in faults per page inspected. Also at a fault detection rate, with fault detection efficiency.
* Execution-based testing is a process of inferring certain behavioral properties of a product based on the results of executing the product in a known environment with selected inputs.
* A simulator is a working model of the environment in which the product executes. Utility is the extent to which a user’s needs are met when a correct product is used under conditions permitted by its specifications.
* Reliability is a measure of the frequency and criticality of product failure.
* Correctness proof is a mathematical technique for showing that a product is correct or that it satisfies its specifications.
* Regression testing is when stored test cases that the product has previously executed correctly must be rerun to ensure that the modifications made to add new functionality to the product have not destroyed the product’s existing functionality.

Chapter 11: Analysis Workflow

* Specification document is a contract between the client and the developer and a solution strategy is a general approach to building the product.
* The analysis workflow aims to obtain a deeper understanding of the requirements, while also describing those requirements in such a way that the resulting design and implementation are easy to maintain.
* Unified Process has three types of classes: Entity classes, boundary classes, and control classes.

1. An entity class models information that is long lived. i.e. account class because information on accounts has to stay in the software product.
2. A boundary class models the interaction between the software product and its actors. Generally associated with an input and output.
3. A control class models complex computations and algorithms.



**Entity Class**

* Entity class extraction consists of three steps:

1. Functional modeling presents scenarios of all the use cases.
2. Entity class modeling determines the entity classes and their attributes. Then determine interrelationships and interactions between entity classes.
3. Dynamic modeling determines the operations performed by or to each entity class or subclass.

* Noun-extraction method can extract candidate entity classes. Identify the nouns and then use these nouns as candidate entity classes. If the noun lies outside the problem boundary, or are abstract nouns, meaning they identify things that have no physical existence, then they can be ignored.

**Boundary and Control Classes**

* Boundary classes are easier to extract. Each input screen, output screen, and printed report is modeled by its own boundary class. Control classes are equally as easy, as each non-trivial computation is modeled by a control class.
* Sequence diagrams can be made and typically depict the objects and messages sent between them in the realization of the flow of events.
* Analysis workflow is an iterative process.

Chapter 12: Design Workflow

* The input to the design workflow is the analysis workflow artifacts.
* In operation-oriented design, the emphasis is on the operations, unlike the data-oriented design, which obviously focuses on the data first.
* The Unified Process is object-oriented design, which aims to design the product in terms of objects, that is, instantiations of the classes and subclasses extracted during the analysis workflow.
* Each package consists of a set of related classes, usually of relevance to a small subset of the actors, that can be implemented as a single unit. The fan-in of a class can be defined as the number of flows into the class plus the number of global data structures accessed by the class. The fan-out similarly is the number of flows out of the class plus the number of global data structures updated by the class.

Chapter 13: Implementation Workflow

* Good programming practices are vital for object-oriented programming, using things such as meaningful, consistent, variable names, self-documenting code, prologue comments, and proper nested if statements.
* One approach to integration is testing each code artifact separately, linking together all 13 code artifacts, and then testing the product as a whole.
* In top-down integration, the code above is implemented and the code below is integrated. Logic artifacts essentially incorporate the decision-making flow of control aspects of the product. They are generally found near the root. Operational artifacts are the opposite, as they perform actual operations of the product. They are generally found toward the bottom, near the leaves. Top-down has a weakness in that potentially reusable code artifacts may not be adequately tested. This leads to defensive-programming, which is a type of design where the calling artifact includes a safety check.
* In bottom-up integration, the artifact below is implemented and integrated before the artifact above gets called. The problem with this is that major design faults are detected late in the implementation workflow, which can be dangerous.
* Sandwich integration is a mix between to the two. Half are integrated top down and the other half are integrated bottom up and the solution is to partition them. Major design faults can be caught early and they are thoroughly tested.
* The aim of implementation workflow is to implement the target software product in the selected implementation language. To do this, large software product is partitioned into smaller subsystems, which are then implemented in parallel by coding teams. These subsystems consist of code artifacts. As soon as a code artifact has been coded, the programmer tests it; this is termed unit testing.
* When testing to specifications, the code is treated as a totally opaque black box. Testing to specifications means the code is ignored and the only information used in drawing up test cases is the specification document. When it is tested to code, they need to see inside the box, hence the term glass-box testing. When testing to code ignores the specification document when selecting test cases. White-box testing is a retarded way of saying glass-box testing.
* In black-box unit testing, there is a type of test called equivalence testing, which is a set of test cases such that any one member of the class is as good a test case as any other. To be more precise, the specified range of numbers of records tha the product must be able to handle defines three equivalence classes: less than 1 record, 1 through 16,383 records, more than that.
* Functional testing is an alternate form of black-box testing in which the methods implemented in the code artifact under test are identified and test data are devised to test each method separately.
* In glass-box techniques, test cases are selected on the basis of examination of the code rather than the specification. The simplest form of testing is statement coverage, which is a running series of test cases during which every statement is executed at least once. There are certain tools to keep track of which statements are still to be executed and a weakness to this is that there is no guarantee that all outcomes of branches are properly tested. Branch coverage is a running a series of tests to ensure that all branches are tested at least once. Again, a tool can be used to keep track. These are called structural tests.
* The most powerful structural test is path coverage, which is testing all paths. Because of loops, this can grow very large and tests have been done to lessen the amount of testing in paths. To reduce the number of paths, two methods can be used, such as linear code sequences and all-definition-use-path coverage.
* Dead code means there is no feasible path in the code artifact and can’t be executed. Computer scientists have developed a number of metrics of software complexity as an aid in determining which code artifacts are most likely to have faults. Such a simple metric for predicting numbers of faults is lines of code.
* Debugging is the detection of the fault and correction of the code. In integration testing, the key point is to first test the new code artifact and then to check that the rest of the partial product continues to behave as it did before the new code artifact was integrated into it. When the integration process is complete, this is termed product testing. When the developers are confident about the correctness o every aspect of the product, it is handed over to the client for acceptance testing.
* In product testing, the product must be subject to stress testing, which is making sure that it behaves correctly when operating under a peak load. Also subject to volume testing, which is making sure it can handle large input files.
* In acceptance testing, it must be performed on actual data rather than on test date.