Chapter 5: Tools of the Trade

* Stepwise Refinement: a means to postpone decisions on details until as late as possible to concentrate on the important issues.
* Without metrics (measurements), it is impossible to detect problems early in the software process, before they get out of hand. [Size, Cost, Duration, Effort, Quality]
* CASE = computer-aided software engineering. Can do things like write user manual, perform integration testing, etc. Tool is a product that assists in just one aspect of the creation of software. UpperCASE = front end, lowerCASE = back end. Tool = one aspect, workbench = series of aspects, environment = complete aspect.
* Programmers need accurate project documents, online help, online programming standards, online manuals, general purpose tools, coding tools, etc.
* Solution to Online Interface Checkers are by incorporating an operating system front-end into the structure editor. Simple programming environment = pretty printer, operating system front-end, online documentation, online interface checking capabilities, etc.
* Two types of software versions: Revisions (written to replace its predecessor) and Variations (designed to coexist).
* In a revision, this can be constructed to fix a fault in the artifact. Cannot throw away an incorrect version, perfect and adaptive maintenance also result in revisions.
* In variations, they are created to coexist in parallel.
* Configuration Control: Every code artifact exists in three forms: source code, compiled code, and executable load image. Configuration is a version of each artifact from which a given version of a product is built. To fix that retarded sentence from the book, I believe that means each one of these code artifacts are created from a previous built code artifact. I guess, in other words, a revision.
* A maintenance manager must set up a baseline, which is a configuration (set of versions) of all the artifacts in the product. Also needs to create private workspaces for each configuration.
* While an artifact is being coded, which the programmer performs informal unit testing, then the artifact is given to the SQA group for methodical testing. An artifact must be subject to configuration control from the time it is passed by SQA.

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 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.

Chapter 14: Postdelivery Maintenance

* Any change to any component of the product (including documentation) after it has passed the acceptance test. This is Postdelivery Maintenance. Some computer scientist nerds prefer to use the term ‘evolution’ to indicate that a product evolves over time.
* The difference between development and maintenance is that it is easier to create a new version than to modify an existing version. Though, economic considerations make maintenance far preferable to redevelopment.
* Corrective Maintenance: to correct residual faults (analysis, design, implementation, documentation, or any other type of faults).
* Perfective Maintenance: client requests changes to improve product effectiveness (add additional functionality, make product run faster, improve maintainability)
* Adaptive Maintenance: Responses to changes in the environment in which the product operates. Not requested by client but imposed externally on the client.
* [Corrective] Suppose there is a defect report handed to the maintenance programmer. They must find this fault with only the source code and nothing else. They must have superb debugging skills to find it. Suppose they find it, they need to fix it without introducing a regression fault. Minimize regression faults: consult the detailed documentation for the product as a whole or consult the detailed documentation for each individual module. If no documentation is found, or it is faulty, then the programmer must change source code. Testing must be done to ensure modification works correctly with special test cases, check for regression faults using stored test data, special test cases for future regression testing, and document all changes.
* [Adaptive/Perfective] Maintenance programmer must go through each workflow individually, using the existing product as a starting point. When programs are developed, specs/designs/code are produced by analysis/design/programming experts. Maintenance programmer must be expert in testing/documentation.
* Defect Reports: If the product appears to function incorrectly, the user files a defect report. It must include enough information to enable the maintenance programmer to recreate the problem. The maintenance programmer should first consult the defect report file. It contains all reported defects not yet fixed and suggestions for working around them. If new defect, programmer should try to find the cause and a way to fix it. Should also be filed in the defect report file.
* Cheaper to make multiple changes to sites, so they should be done in varying updates.
* [Corrective] Assign a maintenance programmer to determine the fault and its cause, then repair it. Test the fix, test the product as a whole (regression testing), update the documentation to reflect the changes made, then update the prologue comments to reflect all changes.
* [Adaptive/Perfective Maintenance] As with corrective maintenance, except there is no defect report. Change in requirements instead.
* Smarter to plan for maintenance over the entire life cycle. Design workflow (use information-hiding techniques), Implementation workflow (select variable names meaningful to future maintenance programmers), Documentation (complete and correct, reflecting the current version of every artifact).
* Reverse engineering is starting with the source code and attempting to re-create the design documents or even the specifications. Going the other way around is forward engineering.
* Defect-tracking tools determine the current status of every report defect.

Chapter 15: More on UML

* UML, or Unified Model Language, is a language used to express ideas. It is the notation, not a methodology.
* A class diagram depicts classes and their interrelationships.
* bank account : Bank Account Class

bank account is an object, an instance of a class Bank Account Class. Underlining denotes an object, colon denotes “an instance of”, boldface and initial upper case letters n Bank Account Class denote that this is a class.

* + indiciates an attribute or operation is public. – denotes attribute or operation is private. # = protected.
* Open diamonds = aggregation. “consists of”. Multiplicity numbers are placed next to the line to show how many each class contains. \* = …or more.
* Composition = stronger than aggregation and is filled in diamond.
* Use case diagram is a set of use cases.