ST Week 12 Notes

L1: Regression Testing

- Post release, a software is *maintained* by developers.
  - Maintenance involves making modifications to code in use, adding new code, removing code etc.
  - These changes are done in response to *change requests* by users/customers. They could be for new features, fixing bugs etc.

Regression testing is the process of validating modified software to detect whether new errors have been introduced into previously tested code and to provide confidence that modifications are correct.

- It is a black-box testing technique.
- Typically very expensive and well used.

Let P be a procedure or program. Let P' be a modified version of P and let T be a test suite for P. A typical regression test proceeds as follows:

1. Select $T' \subseteq T$, a set of test cases to execute on P'.
2. Test P' with T', establishing P''s correctness with respect to T'.
3. If necessary, create T'', a set of new functional or structural test cases for P'.
4. Test P' with T'', establishing P''s correctness with respect to T''.
5. Create T''', a new test suite and test execution profile for P', from T, T' and T''.

Each of the steps in regression testing leads to an important problem to solve.

- Step 1 involves the regression test selection problem: the problem of selecting a subset T' of T with which to test P'.
- Step 3 addresses the coverage identification problem: the problem of identifying portions of P' or its specification that require additional testing.
- Steps 2 and 4 address the test suite execution problem: the problem of efficiently executing test suites and checking test results for correctness.
- Step 5 addresses the test suite maintenance problem: the problem of updating and storing test information.

- Regression test selection is applicable both in cases where the specifications have not changed, and where they have changed.
- In the latter case, it is necessary to identify the test cases in T that are obsolete for P' prior to performing test case selection.
  - Test case t is obsolete for program P' iff t specifies an input to P' that is invalid for P', or t specifies an invalid input-output relation for P'.
- Having identified these test cases and removed them from T, regression test selection can be performed on the remaining test cases.

Selecting regression tests

- Requires knowledge about the system and how it affects by the existing functionalities.
- Tests are selected based on the area of frequent defects.
- Tests are selected to include the area, which has undergone code changes many a times.
- Tests are selected based on the criticality of the features.

Minimization test selection techniques for regression testing attempt to select minimal sets of test cases from T that yield coverage of modified or affected portions of P.

- The problem, in its generality, is undecidable.
- 0-1 integer programming based techniques are available for procedures that are basically a sequence of statements with single entry and single exit (no branching, looping etc.).

Data-flow coverage based regression test selection techniques select test cases that exercise data interactions that have been affected by modifications.

- Techniques that are *not safe* can fail to select a test case that would have revealed a fault in the modified program.
- In contrast, when an explicit set of *safety conditions* can be satisfied, safe regression test selection techniques guarantee that the selected subset, T', contains all test cases in the original test suite T that can reveal faults in P'.
- In this case, a randomly selected set of test cases for P are executed on P'.
- This is often useless and done only to show that some regression testing is done.
- The retest-all technique simply reuses all existing test cases.
- To test P', the technique effectively *selects* all test cases in T.
- Different regression test selection techniques create different trade-offs between the costs of selecting and executing test cases, and the need to achieve sufficient fault detection ability.
- It has been found through empirical studies that the choice of regression test selection algorithm significantly affects the cost-effectiveness of regression testing.
- Most of the testing efforts in regression testing can be automated.
- Typically the previously executed test cases or the selected test cases are again run on a new build.
- Most of the regression test tools are record and playback type.
- The test cases are recorded by navigating through the program under test and verified whether expected results are coming or not.

L2: Software Quality Metrics

Software quality is the capability of a software product to conform to its requirements.

- Several different standards define software quality in different terms.
- It is known that the cost of poor quality is very high, hence there can be no compromise on quality.

In Software Engineering, software quality can be one of the following:

- Software functional quality: Reflects how well it complies with or conforms to a given design, based on functional requirements or specifications.
  - It is the degree to which the correct software was produced.
- Software structural quality: Refers to how it meets non-functional requirements that support the delivery of the functional requirements, such as robustness or maintainability.
  - It is the degree to which a software works as needed.
- Software Quality Assurance: A set of activities to ensure the quality in software engineering processes that ultimately result in quality software products.
  - The activities establish and evaluate the processes that produce products.
  - Involves process-focused action.
- Software Quality Control: A set of activities to ensure the quality in software products.
  - The activities focus on determining the defects in the actual products produced.
  - Involves product-focused action.

A software metric is a measure of the degree/extent to which a software system or process possesses some property. Software metrics are useful in providing quantifiable measurements to various software development processes and the resulting product (code).

- Product metrics: Describe the various characteristics of the software (product).
  - Size, complexity, metrics related to design etc.
- Process metrics: Describe all the characteristics that can be used to improve the development of software and its maintenance.
- Project metrics: Describe the characteristics of the project and its execution.
  - Team size (architects, developers, leads, test engineers etc.), cost, schedule, productivity etc.

Certain metrics can belong to more than one category above.

- Number of lines of code (KLOC)
- Cyclomatic complexity
- Program size (binary)

- Coupling: Coupling is the degree of interdependence between software modules, it is a measure of how closely connected two modules are.
- Cohesion: Cohesion is a measure of the strength of relationship between the methods and data of a class.

A subset of the software metrics that focus on the quality aspects of the software. Typically not associated with project metrics. Three kinds:

- Product quality metrics.
- In-process quality metrics.
- Maintenance quality metrics.

Product Quality Metrics

- Mean time to failure: The time between failures. Used mainly in safety critical systems.
- Defect density: Measures the (number of) defects relative to the size of software (KLOC/function points).
- Issues reported by/related to customers: Number of issues per unit of time (usually month), as reported by customers.
- Satisfaction of customers: Measured through a survey on a five-point scale.

In-Process Quality Metrics

Tracks (the arrival) of defects during formal (machine) testing.

- Defect density.
- Defect arrival pattern: The arrival of defects reported during testing per unit of time interval (week/month), and the backlog of defects over a time period.
- Phase-based defect removal pattern.
- Defect removal effectiveness.

Defect Removal Effectiveness (DRE) is defined as:

$$\text{DRE} = \frac{\text{Defects removed during a development phase}}{\text{Defects latent in the product}} \times 100\%$$

Useful in tracking the defects in each phase of software development. Early removal generally indicates saving in development cost and time.

Maintenance Quality Metrics

- Fix backlog and backlog management.
- Fix response time and fix responsiveness.
- Percent delinquent fixes.
- Fix quality.

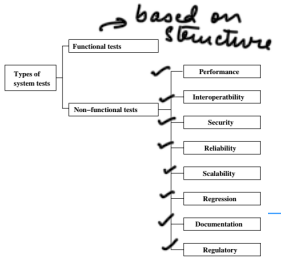
Fix backlog and backlog management is measured using Backlog Management Index (BMI) and is defined as

BMI

$$\text{BMI} = \frac{\text{Number of problems closed during the month}}{\text{Number of problems arrived during the month}} \times 100\%$$

A simple count of the problems that remain unfixed during the end of each time unit (week or month).

L3: Non-Functional Testing



- Interoperability testing determines whether the system can inter-operate with other third-party products.
- Could involve compatibility testing too.
- Compatibility testing involves testing for compatibility with different operating systems, different browsers, different database servers etc.
- Two kinds of compatibility: forward and backward compatibility.

- Security testing determines if a system protects data and maintains security related functionality as intended.
- Aims at testing the following:
  - Confidentiality: Requirement that data and processes by protected from *unauthorized disclosure*.
  - Integrity: Requirement that data and processes be protected from *unauthorized modification*.
  - Availability: Requirement that data and processes be protected from *denial of service to authorized users*.
  - It also deals with authorization and authentication verification.

Types of security testing include the following:

- Verify that only authorized accesses to the system are permitted.
- Verify the correctness of both encryption and decryption algorithms for systems where data/messages are encoded.
- Verify that illegal reading of files, to which the perpetrator is not authorized is not allowed.
- Ensure that virus checkers prevent/curtail entry of viruses into the system.
- Identify *back doors* in the system left open by developers (e.g., buffer overflows). Test for access through back doors.
- Verify different authentication, client-server communication, wireless security protocols etc.

- Reliability tests *measure* the ability of system to keep operating over specified periods (typically several months/years) of time.
- Includes both hardware and software reliability.
- In-depth mathematical analysis techniques are used, a separate area worth delving into.

- Scalability testing verifies that a system can scale up to its engineering limits.
- This involves testing the *limit* of the system, that is, the magnitude of demand that can be placed on the system while continuing to meet its latency, response time and throughput requirements.
- Major causes on limitations to scale up:
  - Data storage limitations
  - Network bandwidth limitations
  - Speed limit (CPU speed)
- Tests are performed by extrapolating basic data used for scalability testing.

- Documentation testing is done to verify the technical accuracy and readability of various documentation including user manuals, tutorials, on-line help etc.
- Usually performed at three levels:
  - Read test: A documentation is reviewed for clarity, organization, flow and accuracy without executing the documented instructions on the system.
  - Hands-on test: On-line help is exercised and the error messages verified to evaluate their accuracy and usefulness.
  - Functional test: Instructions embodied in the documentation are followed to verify that the system works as it has been documented.

Regulatory Testing

- Each country has regulatory bodies guiding the availability of a product in that country.
- CE (Conformite Europeene), CSA (Canadian Standards Association), FCC (Federal Communications Commission) etc. are some of them.
- In addition, safety critical systems have their own regulatory requirements, per, domain.
  - Aerospace: ARP standards, DO standards by RTCA etc.
  - Automotive: IEC 61508, MISRA guidelines etc.
- Exhaustive testing and documentation is done throughout the development to cater to these regulatory standards.

- Performance testing is done to determine the system parameters in terms of *responsiveness* and *stability* under various workloads.
- Performance testing measures the quality attributes of the system, such as scalability, reliability and resource usage.
- For e.g., an expected performance could be "A transaction in an on-line system requires a response of less than 1 second 90% of the time".
- Tests measure the performance of the actual system compared to the expected quality parameters.

Types

- Load testing: It is the simplest form of testing conducted to understand the behaviour of the system under a specific load. Load testing will result in measuring important business critical transactions and load on the database, application server, etc., are monitored.
- Stress testing: It is performed to find the upper limit capacity of the system and also to determine how the system performs if the current load goes well above the expected maximum.
- Soak testing: Soak Testing also known as endurance testing, is performed to determine the system parameters under continuous expected load. During soak tests the parameters such as memory utilization is monitored to detect memory leaks or other performance issues.
- Spike testing: Spike testing is performed by increasing the number of users suddenly by a very large amount and measuring the performance of the system. The main aim is to determine whether the system will be able to sustain the workload.

Eg. Release of big exam results

- Performance testing goals:
  - Demonstrate that the system meets performance criteria.
  - Compare two systems to find which performs better.
  - Identify and measure which parts of the system or workload cause the system to perform badly.
- Performance tests typically always include the following:
  - Number of concurrent users.
  - Throughput or transaction rate.
  - (Server) response time.
  - Any other specific measures.

The following are some of the tools that can create and run performance test scripts.

- HP LoadRunner.
- NeoLoad: For web and mobile applications.
- Apache JMeter: Good for web applications, open source.
- Rational Performance Tester.
- Gatling: Good for web applications, open source.

L4: TDD → Agile process, FAIL → PASS → REFACTOR

- Test-Driven Development (TDD): Write tests and code in small, incremental and alternating steps.
- A test case is written first, there is no code as yet. So, the test case fails.
- Immediately, just enough code is written so that the test case will pass.
- The next set of test cases are written and this process id repeated till all desired functionality is obtained.
- As code size increases, re-factoring is done. All original tests must still pass.

The complete Java code for date validation routine is basically just a composition of all the above, i.e., a *blind* way of just putting them together.
This results in code smells, the resulting code is not efficient, too cumbersome for (human) readability.
The code needs to be re-factored to remove duplicates, re-arranging code segments to make them logically consistent etc.

- Advantages of TDD:
  - Fault isolation is easy – a new fault is most likely to be present in the newly added code.
  - A TDD based project is ideal for pair programming— one for writing code and one for testing.
- Disadvanteges of TDD:
  - Heavy dependence on test frameworks.
  - TDD scaling up to large (number of lines of code/complexity) is still open.

a new agile method, not so popular yet.