OOAD & Software Engineering (UE18CS353)
Unit 4

Aronya Baksy

April 2021

1 Software Testing

- Testing is executing software in a simulated or real environment, using inputs selected somehow
- Inadequate/incomplete testing leads to catastrophes (Therac-25, Ariane-5 etc.)
- In a normal waterfall model, testing activity follows the Requirement, Design, Implementation phases. This causes **large costs** in fixing errors and very late detection due to the big-bang nature of the testing
- In a V model, the testing is planned along with the corresponding SDLC phase. This results in better chance of picking up errors early and fixing them with minimal cost.
- In an Agile approach like Scrum:
  - The entire team takes part in the entire test activity: test planning, test specification, test execution, test evaluation and result reporting
  - Each sprint involves creating unit tests for the code written in that sprint. These unit tests are created and run by the developers.
  - Testers are in charge of testing functional and non functional attributes
  - The customer carries out the User Acceptance test at the end of the sprint. The feedback from this test is used as input to the next sprint.
  - Test results are collected and maintained at the end of each sprint.

1.1 Objectives of Testing

- **Demonstration**: System can be used with acceptable risk level, in special conditions, and is ready for integration
- **Detection**: of defects, errors, and deficiencies. Determine quality of components and the overall system (capabilites and limitations)
- **Prevention**: of errors and defects being propagated to later stages. Clarify non-functional requirements,

1.2 Validation and Verification

Validation	**Verification**
The assurance that a product, service, or system meets the needs of the customer and other identified stakeholders.	The evaluation of whether or not a product, service, or system complies with a regulation, requirement, specification, or imposed condition.
Done via dynamic testing (execution of code)	Done using static testing
Target the capabilities and features defined in the projects scope and requirements definition	Targets on software architecture, design

1.3 Terminologies

- **Defect**: deviation from requirements. Imperfection due to design or coding mistake
- **Bug**: Programmer error that prevents correct operation of the system.
- **Failure**: The state of the system that is caused by a defect. Occurs during development or later, and leads to unfulfillment of the requirements
- **Issue**: Raised by the end user when deployed product does not meet functional or non-functional requirements. Tracked using software like Jira

2 Testing Types

2.1 Black-Box Testing (Functional Testing)

- Software is treated as a black box, no knowledge of internal implementation or logic.
- Objective is to identify invalid outputs when valid or invalid inputs are given to the system.
- Rqquires generated test cases to be passed to tester who executes the test cases on the system.
- **Advantage**: early test planning, best for large units of code
- **Drawbacks**: Path coverage not good, hard to direct tests towards error-prone code

2.2 White-Box Testing (Structural Testing)

- Factor in the logic and structure of the code.
- Test cases derived using the knowledge of the above. Hence test cases are not made until implementation is complete
- **Advantages**: reveals hidden errors, partitioning by equivalence
- **Drawbacks**: testers need to have programming skills, hard to test all code (for large units)

2.3 Grey-Box Testing

- Mix of functional and structural testing.
- Test cases are designed using white-box methodology, but actual testing is done at the user level (black-box method)
- Can be used for integration testing between modules

2.4 Static Testing

- Testing that does not involve any execution of code.
- Carried out early on in the development stage, find errors and correct them as early as possible
- Static testing involves:
  - Evaluation of code quality
  - Data Flow
  - Control Flow
  - Cyclomatic Complexity

2.5 Dynamic Testing

- Involves running code to analyze the dynamic behaviour of the program under test.
- Involves running the program with inputs and analyzing the outputs.
- Can find difficult and complex defects which are not easily detectable by static testing.
- Types of dynamic testing:
  - Testing based on techniques like Code based or Fault based
  - Testing based on how the testing is done (Manual or Automatic)
  - Testing based on different levels of testing (unit, integration, system, acceptance)

2.6 Based on Testing Technique

2.6.1 Control-Flow based Testing

- A control-flow path (aka a control-flow graph) is a representation of the logical structure of the program, in terms of the paths taken by a running program
- Metrics: branch coverage of the test cases

2.6.2 Data-Flow based Testing

- Finds paths in program based on location of definition and use of data variables in the program.
- Finds anomalies such as use without declaration, declaration without use, multiple declaration, deallocation without use etc.

2.6.3 Error based Testing

- Use prior experience and domain knowledge to identify most common faults in the program
- Then design test cases to attack those faults and evaluate.

2.6.4 Fault based Testing

- Introduce faults into the code on purpose
- May lead to other faults being detected. Ideal outcome is that all test cases designed for correct program should fail on this new program.

2.6.5 Mutation Testing

- Select certain statements (could be branch, arithmetic etc.) and make mutations on those.
- Run all the tests against the mutants and the original. All mutants should fail
- Reveals complex faults

2.6.6 Specification based Testing

- Test output given an input strictly according to requirements laid out in the SRS

2.6.7 Intuition based, Usage based, Domain based

- Based on the tester's intuition, the real world use cases and the application domain knowledge to design test cases

2.7 Manual vs Automated Testing

2.7.1 Manual Testing

- Human performs tests without any test scripts or additional helpers
- Used for complex tests where automation is expensive
- Slow, tedious, low coverage

2.7.2 Automated Testing

- Use test automation frameworks and test scripts to run tests with minimal human intervention
- Fast and repeatable but requires extra time for writing test scripts and framework maintenance.
- Efficient for periodic regular tests

3 Testing Levels

3.1 Unit Testing

- Test smallest individually executable units of code. Check for coding or construction errors.
- Unit test focussed on Algos and DS, interfaces, boundary conditions, error handling
- For OO projects this can be at the class level or at the method level

3.2 Integration Testing

- Verify interfaces between components against the high-level design. Done by programmers
- Identify resource contention and timing problems that are not detected by unit tests.
- Integration test strategies:
  - **Big Bang**: integrate all modules and then check if system is running
  - **Top-Down**: Simulate behaviour of lower level modules that are incomplete, using stubs. Integrate high level modules first then work downwards, in a depth-first manner.
  - **Bottom-Up**: Lowest level components are tested first, then used to facilitate the testing of higher level components (repeat until entire hierarchy is covered)

3.3 System Testing

- Verify that a completely integrated system meets the requirements. Fix defects in system as a whole, as well as interfaces b/w components.
- Requirements are defined by an SRS or an FRS (Functional Req. Spec.) document
- Varieties of system testing:
  - Regression testing: test whether changes cause side effects
  - Installation testing
  - Startup/shutdown testing
  - Load/Stress testing
  - Platform testing (for cross-platform apps)
  - Localization testing (does the app work across different locales of the world)
  - Functional and non-functional testing

3.4 Acceptance Testing

- Run test suite on complete system. Each test case in the suite includes a particular operating condition or use case in the real world.
- Test environment is designed to be as close as possible to real world user.
- Created by analysts, test engineers, customers and developers. Vital to include business logic tests as well as UI validation tests.
- In an Agile methodology, an acceptance test case is similar to a single user story being played out.
- Provide confidence that system meets business requirements. Acts as final gateway in terms of release, and contractual obligation between stakeholder and client.
- The software can even be given to a small set of representative users for testing.
  - In α-testing, the group of test users test the product within the confined environment of the development group
  - In β-testing, the group of test users test the product in the outside world.

4 Test Planning and Strategy

- Test planning is the process of evolving a software test plan which discusses what, when and how testing has to be done as part of the project to ensure quality expectations can be met.
- Test planning involves the following 9 activities:

4.1 Define context and scope

- Review use case scenarios, discussions with developers and designers
- Review the documentation, perform a product walk-through
- Scope of testing is decided by customer requirements (domain, risk, quality), budget and schedule constraints, skills available in the test team.

4.2 Define test adequacy criteria

- Criteria that determine when to stop testing i.e. consider testing to be complete for an iteration or sprint.
- Based on code coverage criteria (lines of code, branch coverage etc.) or based on percentage of defects covered.

4.3 Test Strategy

- Test model, test types, test automation, test environment, and risk analysis + contingency planning come under test strategy

4.3.1 Test Models/Mindsets

- **Demonstration Model**: whether the software runs successfully and passes the test cases designed for it. May lead to unconscious bias where only successful cases are tested
- **Evaluation Model**: Focus on analysis and review techniques to detect faults in requirements and design documents
- **Destruction Model**: try to find as many faults in the program as possible. Difficult to decide adequacy because number of remaining faults is never known.
- **Preventive Model**: Prevent faults as early as possible with careful design, planning of test activity.

4.3.2 Test Type Chosen

- Each life cycle phase has testable outcomes
- Start by low-level testing of small components and move towards integration into larger components.

4.3.3 Test Environment

- Software or hardware setup designed for testers to be able to execute their test cases.
- Consists of the application under test, the server OS, test data, Database, front end for testing, servers/storage/network and documentation needed.
- Test environment needs to be maintained and managed, may involve periodic upgrades to keep the environment working.
- Challenges: planning, resource usage, remote environments, setup time and cost, sharing challenges, configuration

4.3.4 Automation Strategy

- Define goals, plan test approach
- Select automation framework and test tool
- Develop test cases, then code test scripts. Execution of test cases

4.3.5 Testing Tools

- Criteria for selection:
  - Compatibility between application under test, and testing tool to be used
  - Proficiency of testers in the selected tool, for best utilization
  - Balance between features offered, test report detail generated, ease of use.
  - Cross-platform support
  - Acceptance/popularity as an indication of available support and documentation
  - Cost

4.3.6 Risk Analysis

- Risks in terms of:
  - Changes in business/technology goals and competition
  - Resource available
  - Quality of product, test types/models not usable
  - Test automation and tool problems

4.4 Deliverable List

- Most commonly, in the form of a list of test cases, and test specifications for each module.

4.5 Test Schedule Creation

- Create a WBS for the testing activites, using estimation techniques such as CoCoMo and others.
- Includes estimation for building test strategy/specification/test cases/test environment setup and test execution, test reporting etc.)

4.6 Planning, Identifying, Allocating Resources

- Test resource allocation between different teams in terms of hardware (servers/network/storage) and software (tools)
- Identifying resources available, number and type of people working on project, and finally test environment and data needed by each team.
- Done in conjunction with the scheduling activity.

4.7 Identify milestones and risks

- Identify milestones in terms of resource allocation and the modified schedule.
- Risks for completion of each of the task from a schedule and quality perspective is identified, analyzed, mitigation plans made and the triggers for kick-off identified.
- Milestones are used to track budget and schedule overruns, as well as identify risk triggers and mitigation plans.

4.8 Identify measures and metrics

- Measures:
  1. The number of test cases planned and created
  2. Number of test cases run
  3. Time spent on creation, execution
  4. the number of errors found
- Metrics:
  1. Test cases executed per day
  2. Issues per kLoC, crititcal issues per kLoC
  3. Number of requirements traced over the lifecycle

5 Test Process

- **Planning and control**: test strategy, adequacy criteria, schedule, review+approval boards
- **Analysis and Design**: Test requirements and product architecture
- **Implementation and execution**: design and run test cases, collect metrics and log results
- **Evaluate exit criteria and report**: Set up test stopping criteria
- **Closure**: Verify all planned activities done, archive all config items created during testing

6 Roles in testing

- **Director**: Oversight, co-ordination, high level vision and primary contact for clients and stakeholders to connect
- **Test Manager**: Prepare test plan and test strategy, monitor and control the test process
- **Infra Manager**: Manage all test infrastructure, capacity/maintenance and configuration support
- **Automation Manager**: Manage development/selection of testing tools and scripts
- **Test Architect**: Design test infrastructure, select/drive requirements for testing tools, validate the test strategy.

- **Test Analyst**: Maps customer environment and requirements to test environment and documentation
- **Test Engineer**: Carries out tests according to the methodology, plan and strategy developed.
- **Test Development Engineer**: Develops scripts, tools for testing

7 Test Metrics and Measurements

- Metric: a quantitative measure of the testing process indicating the progress, quality, productivity and the degree to which a system, possesses a given attribute.
- Metrics must be understandable, quantitative, applicable, repeatable, economical to compute and language independent.

7.1 Test Measures

- Size in LOC
- Fault density: bug per kLoC
- MTBF and its inverse, failure rate
- Defect distribution across SDLC phases, and across modules
- Defect leakage:

$$Leakage = \frac{\#\text{ of defects found in UAT}}{\#\text{ of defects found before UAT}} \times 100$$

- Test coverage: statement, branch, condition, loop, path, data flow coverage etc.
- Percent of tests executed and defects completed.
- Defect discovery rate
- Percent injected faults discovered, mutation score: percent of mutants killed per total number of mutations introduced.

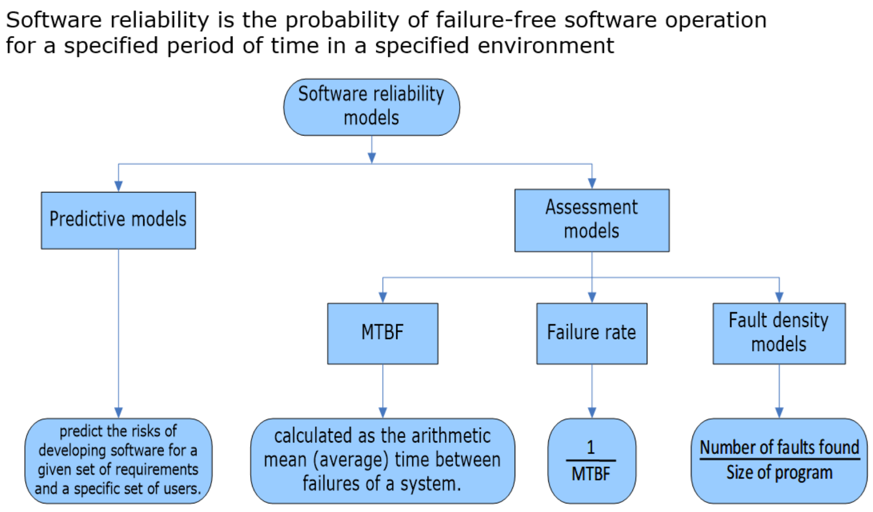
Software reliability is the probability of failure-free software operation for a specified period of time in a specified environment
Software reliability models
Predictive models
Assessment models
MTBF
Failure rate
Fault density models
predict the risks of developing software for a given set of requirements and a specific set of users.
calculated as the arithmetic mean (average) time between failures of a system.
1 / MTBF
Number of faults found / Size of program

Figure 1: Test Reliability

8 Software Maintenance

- The sustaining process of modifying a software system or component after delivery to correct faults, improve performance or other attributes, or adapt to a changed environment
- Fixing defects from real-world use, fixing requirement and design defects, feature and performance enhancements, reaction to environmental changes, interfacing with other systems and retiring legacy components.
- Maintenance activities:
  - Control over daily functioning, modifications to software
  - Maintain performance at acceptable level
  - Bug removal from existing software
- Maintenance Drivers:
  - Understanding of product from documentation
  - Look at present and the change request at hand, to identify how to satisfy the task at hand
  - Analyze the consequences of the solution as to how it affects any other future maintenance

8.1 Types

- **Predictive**: Modification after delivery to detect latent faults and correct them before they are exposed in production
- **Perfective**: Modification after delivery to improve performance or maintainability
- **Corrective**: Correction of detected faults post delivery
- **Adaptive**: Keep a product usable after delivery in response to changing environment

8.2 Issues in maintenance

- **Technical**: caused by limitations in code/documentation quality, limited understanding of the project and lack of well-defined change scope
- **Management**: Maintain people with high skill, align with economic objectives (due to lack of direct RoI from maintenance)
- **Outsourcing** of maintenance leads to challenges like IP protection, lack of control over process of development, learning curve
- **Costs** of staff availability, lifetime of project, contracts
- **Predictability**: schedule, time alloted for maintenance.

8.3 Maintenance Activities

8.3.1 Process Implementation

- Develop, document, and execute maintenance process plans including usage of tools as part of the maintenance process
- Includes procedures for handling user reports and modification requests, issue tracking, response to user

8.3.2 SMLC

- Identify problem
- Analyze the problem
- Open formal modification request and get it approved
- Design the change (modification to documentation and design)
- Implement the change
- Unit-test the change
- System test, regression test, acceptance test
- Delivery of the changed product as a patch.

8.3.3 Migration

- From old version to patched version of a product
- Migration plan involves:
  - Notifying the user
  - Applying the patch using restart or no restart (depends on kind of patch)
  - White papers/tools used for more elaborate products
  - Verify the process, support old data based on agreements

8.3.4 Retirement

- Retirement planning involves a timeline and methods for maintaining old data as needed
- Dispose ageing hardware and associated licenses and contracts

8.4 Reverse Engineering and re-engineering

- **Reverse engineering** generates an alternate representation of the software apart from the existing code and documentation.
- Allows maintainers to diagnose faults in components/their interfaces, without modifying/creating any product.
- **Re-engineering** is hte process of modifying software to make it more understandable, and extensible.
- Done only when standard maintenance techniques are not feasible or economical anymore.
- May include refactoring, expected to improve CPU + memory untilization, and readability.

9 Software Quality

- Quality either in terms of actual product quality, or process quality. Improving product quality can mean improving either one
- Relevant standards: for products - ISO 9126, for processes - ISO 9001
- Quality from **operation** perspective:
  - Correctness
  - Reliability
  - Efficiency
  - Integrity (security)

  - Usability
  - Functionality
  - Availability

- Quality from **maintenance** perspective
  - Maintainability
  - Testability
  - Flexibility (changes)

- Quality from **transition** perspective
  - Portability
  - Reusability
  - Interoperability (with other systems)

- Quality from **environment** perspective
  - Responsiveness
  - Predictability
  - Productivity (throughput)
  - Customer and employee satisfaction

- **FLURPS Model** for quality in terms of functional and non-functional attributes: Functionality, Localization, Usability, Reliability, Performance, Support

9.1 Software QA

- Means to monitor SE process, maintain quality throughout the same
- Done by
  - **Managers**: establish plans, oversight, processes, methods
  - **Developers**: apply valid methods, reviews, planned testing
  - **SQA Group**: Record keeping, analysis, report, customer representation
- SQA Plan consists of:
  - Responsibility management
  - Document management and control
  - Requirement scope
  - Design and development control
  - Testing and QA, Auditing procedure
  - Risk mitigation, defect management