Software Testing

Week-1

L1 Motivation

- graphs, logic, syntax - based testing — reg. testing
- Software is ubiquitous - offer good performance in terms of response time, performance & no errors.
- Eg. Ariane 5 (Rocket), Therac 25 (Cancer Treatment), Intel Pentium Bug
- Testing is the most impt. technique to find & eliminate errors in software. Takes up 50% efforts in SDLC.
- Types of softwares - embedded safety critical, enterprise, web & mobile applicⁿs, etc.
- cost of late testing is too high.
- Agile ensures that testing becomes the integral part of every phase of SDLC.
- Testing is used to find errors in the software, doesn't prove that the software is correct. It can't be fully automated, needs human interveⁿ. It shouldn't be replaced by reviews, inspections, & audits, etc.

L2 SDLC

- term used by the industry to define a process for designing, dev., testing & maintaining a high quality prod
- Planning → Design → Coding → Testing → Maintenance
- Planning - detailed req., analysis, products - req. doc., project plan.
- Design - details about modules, prod → design & architecture doc. & models
- Development - Coding, unit testing, debugging, prod → code & documentⁿ, test cases
- Testing - test, accreditions, certificⁿ, prod → Test cases & test documentⁿ

- Maintenance - post deployment, add features, fix errors, regression testing
- SDLC Models - V, Waterfall, spiral, incremental, RAD, agile, etc.
- V-Model - Focus on verificⁿ & validaⁿ, follows waterfall model. Each phase has direct mapping to a corresponding testing phase.
- Agile - most widely used, adaptive & focus of fast delivery of features. All steps of SDLC are repeated in incremental iteraⁿˢ. Extensive customer interacⁿˢ, quick delivery & rapid response to Δ in req.
- Proj. Mgmt. & Team Mgmt.
- Traceability Matrix - Doc. that links each (artifacts of dev. phase to those of other phases
  → Key to effective testing

L3 Terminologies

- ST - is the act of examining the artifacts & the behaviour of the software under test by validaⁿ & verificaⁿ.
- Validation → process of evaluating software at the end of SD to ensure compliance with intended usage, checking software meet its req
- Verificaⁿ → process of determining whether the prod of given phase of SD process fulfill the req established at the start of that phase.
- Formal methods → model checking, theorem proving, program analysis

- modelling & stimula", software quality assurance, accredita", test-driven dev.
- Fault → static defect. Missing func" or wrong func"
- Failure → An ext, incorrect behaviour w.r.t to req. or other descrip" of the exp behaviour.
- A test case involves inputs to the software & exp. outputs: A failed test case indicates an error.
- Unit - done by dev. during coding
- Integra" - comp put together & tested..
- System - full sys. implementa" & on running platform.
- Acceptance - done by end customers to ensure all the req. are delivered.
- Beta - Beta version, by end users, after release.
- Functional - done to ensure software meets the func.
- Stress - how sys. behaves in peak/infav. conditions
- Performance - ensures speed & response time of sys.
- Usability - evaluates UI & aesthetics
- Regression - ensures modifica"n is correctly working

Black Box	White Box
examines software w/o looking into code. unit, integra", sys., accept, beta, usability, stress, etc.	tests the int. struc. of the code. unit, integra" & sys.

- Testing activities → design, automation, execut", evalua"
  - adj. → test mgnt, test maintainence & test documenta"
- Test design - most critical job, create effective test cases, can't be automated, need basic knowledge of algos., maths, domain of the software.
- Test automation - convert test cases into executable scripts
  - 2 impt. → Observability & Controllability, specific to test execution framework

- test execution - run tests on software, record results, can be fully automated, use JUnit
- test evaluation - evaluate testing results, report identified errors, isolate faults, need domain knowledge & exp.

L4 Terminologies & Processes

- level-0 → no diff. btw. testing & debugging
- level-1 → purpose is to show correctness
- level-2 → purpose is to software doesn't work
- level-3 → purpose is not to prove anything, but to reduce the risk of using the software.
- level-4 → testing is a mental discipline that helps all IT prof. develop higher quality software.
- we will deal with level 3 & 4 thinking, write test objectives, plan & achieve coverage in code.
- Controllability - It is about how easy it is to provide inputs to the software module under test in terms of reaching the module & running the test cases on the module under test.
- Observability - It is about how easy it is to observe the software module under test & check if the module behaved as expected.

15 JUnit as an Example

- test automa^n is the 2nd step of test cases. It should meet the RIPR model requirements. Need to give prefix values towards reachability & postfix values to ensure propaga^n.
- test automa^n is the process of controlling the execu^n of tests, the comparison of actual outcomes to expected outcomes, the setting up of test preconditions, & other test control & test reporting fun^{ns}.
- Prefix values - Inputs necessary to put the software into the appropriate state to receive the test case values.
- Postfix values - Any inputs that need to be sent to the software after the test case values to sent.
  - ↳ Verifican^n values - values needed to see the results of the test case values.
  - ↳ Exit values - Values or commands needed to terminat the program or otherwise return it to a stable state.
- A test case ready for execu^n has test case values, prefix, postfix & expected results necessary for a comp. execu^n & evalua^n of the software artifact under test.
- Executable test script - A test case that is prepared in a form to be executed automatically on the test software & produce a report.
- JUnit: org.
- Assertions for testing exp. results. Test features for sharing common test data. Test suites for easily org. & running tests. Graphical & textual test runners.
- They should be initialized in a @BeforeEach/@BeforeAll method.
- Can be deallocated or result in an @AfterEach/@AfterAll

- Data-driven tests - It calls a constructor for each collec^{n} of test values. Same tests are then run on each set of data values. Collec^{n} of data values defined by method tagged with @Params annotat^{n}.