2. TestingThroughouttheSoftwareDevelopmentLifecycle – 130 minutes

Keywords

acceptancetesting,black-boxtesting,componentintegrationtesting,componenttesting,confirmation testing, functional testing, integration testing, maintenance testing, non-functional testing, regression testing,shift-left,systemintegrationtesting,systemtesting,testlevel, testobject,testtype, white-box testing

LearningObjectivesforChapter 2:

2.1 TestingintheContextof aSoftwareDevelopment Lifecycle

FL-2.1.1 (K2) Explain the impact of the chosen software development lifecycle on testing

FL-2.1.2 (K1)Recallgoodtestingpracticesthatapplytoallsoftwaredevelopment lifecycles FL-

2.1.3 (K1) Recall the examples of test-first approaches to development

FL-2.1.4 (K2)SummarizehowDevOpsmighthaveanimpactontesting FL-

2.1.5 (K2) Explain the shift-left approach

FL-2.1.6 (K2)Explainhowretrospectivescanbeusedasamechanismforprocessimprovement

2.2 TestLevelsandTestTypes

FL-2.2.1 (K2)Distinguishthedifferenttestlevels

FL-2.2.2 (K2)Distinguishthedifferenttesttypes

FL-2.2.3 (K2)Distinguishconfirmationtestingfromregressiontesting

2.3 MaintenanceTesting

FL-2.3.1 (K2)Summarizemaintenancetestinganditstriggers

s

2.1. TestingintheContextofaSoftwareDevelopmentLifecycle

A software development lifecycle (SDLC) model is an abstract, high-level representation of the software development process. A SDLC model defines how different development phases and types of activities performedwithinthisprocess relate toeachother,bothlogicallyandchronologically.ExamplesofSDLC models include: sequential development models (e.g., waterfall model, V-model), iterative development models (e.g., spiral model, prototyping), and incremental development models (e.g., Unified Process).

Someactivitieswithinsoftwaredevelopmentprocessescanalsobedescribed bymoredetailedsoftware development methods and Agile practices. Examples include: acceptance test-driven development (ATDD),behavior-driven development(BDD),domain-drivendesign (DDD),extremeprogramming (XP), feature-driven development (FDD), Kanban, Lean IT, Scrum, and test-driven development (TDD).

2.1.1. ImpactoftheSoftwareDevelopmentLifecycleonTesting

Testingmustbeadaptedtothe SDLCtosucceed.ThechoiceoftheSDLCimpactson the:

- Scopeandtimingoftest activities(e.g.,testlevelsandtesttypes)
- Levelofdetailoftestdocumentation
- Choiceoftesttechniquesandtest approach
- Extentoftestautomation
- Roleandresponsibilitiesofa tester

In sequential development models, in the initial phases testers typically participate in requirement reviews,test analysis,andtest design.Theexecutablecodeisusuallycreatedinthelaterphases,so typically dynamic testing cannot be performed early in the SDLC.

In some iterative and incremental development models, it is assumed that each iteration delivers a workingprototypeor productincrement.Thisimpliesthat ineachiteration bothstaticanddynamictesting maybe performedatalltestlevels. Frequent delivery ofincrementsrequires fastfeedback and extensive regression testing.

Agile software development assumes that change may occur throughout the project. Therefore, lightweightworkproductdocumentationandextensivetestautomationtomakeregressiontestingeasier are favored in agile projects. Also, most of the manual testing tends to be done using experience-based test techniques (see Section 4.4) that do not require extensive prior test analysis and design.

2.1.2. SoftwareDevelopmentLifecycleandGoodTestingPractices

Goodtestingpractices,independentofthechosenSDLCmodel,includethefollowing:

- Foreverysoftwaredevelopmentactivity,thereisacorrespondingtestactivity, sothatall development activities are subject to quality control
- Differenttestlevels (seechapter2.2.1)havespecific anddifferenttestobjectives,whichallows for testing to be appropriately comprehensive while avoiding redundancy

- Test analysis and design for a given test level begins during the corresponding development phaseof theSDLC,so that testingcanadheretotheprincipleofearlytesting(see section 1.3)

- Testers are involved in reviewing work products as soon as drafts of this documentation are available,so thatthisearliertestinganddefectdetection cansupporttheshift-leftstrategy (see section 2.1.5)

2.1.3. TestingasaDriverforSoftware Development

TDD, ATDD and BDD are similar development approaches, where tests are defined as a means of directingdevelopment.Eachoftheseapproachesimplementstheprincipleofearlytesting(seesection 1.3) and follows a shift-left approach (see section 2.1.5), since the tests are defined before the code is written.Theysupport aniterative development model.Theseapproachesarecharacterizedasfollows:

Test-DrivenDevelopment(TDD):

- Directsthecodingthroughtestcases(insteadofextensivesoftwaredesign)(Beck2003)
- Testsarewrittenfirst, thenthecodeiswrittentosatisfythetests,andthenthe testsandcodeare refactored

AcceptanceTest-DrivenDevelopment(ATDD)(seesection4.5.3):

- Derivestestsfromacceptancecriteriaas partofthesystemdesignprocess (Gärtner2011)
- Testsarewrittenbeforethepartof theapplication isdevelopedtosatisfythetests

Behavior-Driven Development (BDD):

- Expressesthe desiredbehaviorofan applicationwithtest caseswritteninasimpleformof natural language, which is easy to understand by stakeholders – usually using the Given/When/Then format. (Chelimsky 2010)
- Testcasesarethenautomaticallytranslatedintoexecutabletests

Forallthe aboveapproaches,testsmay persist asautomatedtests toensure thecodequalityinfuture adaptions / refactoring.

2.1.4. DevOpsand Testing

DevOps is an organizational approach aiming to create synergy by getting development (including testing)andoperationstoworktogethertoachieveasetofcommongoals.DevOps requiresacultural shiftwithinan organization tobridgethegapsbetweendevelopment (includingtesting)andoperations while treating their functions with equal value. DevOps promotes team autonomy, fast feedback, integrated toolchains, and technical practices like continuous integration (CI) and continuous delivery (CD). This enables the teams to build, test and release high-quality code faster through a DevOps delivery pipeline (Kim 2016).

FromthetestingperspectiveSomeofthebenefitsofDevOps are:

- Fastfeedbackonthecodequality,andwhetherchangesadverselyaffectexistingcode
- CIpromotesashift-leftapproachintesting(seesection2.1.5)by encouragingdevelopers to submit high quality code accompanied by component tests and static analysis

- PromotesautomatedprocesseslikeCI/CDthatfacilitateestablishingstabletestenvironments
- Increasestheviewonnon-functionalqualitycharacteristics(e.g.,performance,reliability)

- Automationthroughadeliverypipelinereducestheneedforrepetitivemanualtesting
- Theriskinregressionis minimizedduetothescaleandrangeof automatedregressiontests

DevOps is not without its risks and challenges, which include:

- TheDevOpsdeliverypipelinemustbedefinedandestablished
- CI/CDtoolsmustbeintroducedandmaintained
- Testautomationrequiresadditionalresourcesandmaybedifficulttoestablishandmaintain

AlthoughDevOpscomes withahighlevel ofautomatedtesting,manualtesting – especiallyfromthe user's perspective – will still be needed.

2.1.5. Shift-LeftApproach

The principle of early testing (see section 1.3) is sometimes referred to as shift-left because it is an approachwheretestingisperformed earlierinthe SDLC. Shift-leftnormallysuggeststhattestingshould be done earlier (e.g., not waiting for code to be implemented or for components to be integrated), but it does not mean that testing later in the SDLC should be neglected.

Therearesome goodpracticesthat illustratehowtoachievea“shift-left”intesting,whichinclude:

- Reviewingthespecificationfromtheperspectiveoftesting.Thesereviewactivitieson specifications often find potential defects, such as ambiguities, incompleteness, and inconsistencies
- Writingtestcasesbeforethecodeiswrittenandhave the coderunina testharness duringcode implementation
- UsingCIand evenbetter CDasitcomeswithfastfeedbackandautomatedcomponentteststo accompany source code when it is submitted to the code repository
- Completingstaticanalysisofsourcecodepriortodynamictesting,oraspart ofan automated process
- Performing non-functional testing starting at the component test level, where possible. This is a formof shift-leftasthesenon-functional testtypestendto beperformedlaterintheSDLCwhen a complete system and a representative test environment are available

Ashift-leftapproachmightresultinextratraining, effortand/orcostsearlierintheprocessbutisexpected to save efforts and/or costs later in the process.

Fortheshift-leftapproachitisimportantthatstakeholders areconvinced andbought intothis concept.

2.1.6. RetrospectivesandProcessImprovement

Retrospectives (also known as “post-project meetings” and project retrospectives) are often held at the end of a project or an iteration, at a release milestone, or can be held when needed. The timing and organization of the retrospectives depend on the particular SDLC model being followed. In these

meetings theparticipants(notonlytesters, butalsoe.g.,developers,architects,productowner,business analysts) discuss:

- Whatwassuccessful,andshouldberetained?

s

- Whatwasnotsuccessful andcouldbeimproved?
- Howtoincorporatetheimprovementsandretainthesuccessesinthefuture?

Theresultsshouldbe recorded and are normallypartofthetestcompletionreport (seesection5.3.2). Retrospectives are critical for the successful implementation of continuous improvement and it is important that any recommended improvements are followed up.

Typicalbenefitsfortestinginclude:

- Increasedtesteffectiveness/efficiency(e.g.,byimplementingsuggestionsforprocess improvement)
- Increasedqualityoftestware(e.g.,byjointlyreviewingthetestprocesses)
- Teambondingandlearning(e.g., asaresultof theopportunitytoraiseissues andpropose improvement points)
- Improvedqualityofthetest basis(e.g.,as deficiencies in theextentandquality ofthe requirements could be addressed and solved)
- Bettercooperationbetweendevelopmentandtesting (e.g.,ascollaborationisreviewedand optimized regularly)

2.2. TestLevelsandTestTypes

Testlevels aregroupsoftestactivitiesthatareorganized andmanagedtogether.Eachtestlevelisan instance of the test process, performed in relation to software at a given stage of development, from individual components to complete systems or, where applicable, systems of systems.

Testlevels are relatedtoother activitieswithintheSDLC. InsequentialSDLCmodels,thetestlevelsare oftendefinedsuchthatthe exitcriteriaof onelevelarepartoftheentrycriteriaforthe nextlevel.Insome iterative models, this may not apply. Development activities may span through multiple test levels. Test levels may overlap in time.

Testtypesaregroupsoftestactivities relatedtospecific qualitycharacteristics and mostof thosetest activities can be performed at every test level.

2.2.1. TestLevels

Inthissyllabus, thefollowingfivetestlevelsaredescribed:

- **Componenttesting**(alsoknownasunittesting)focusesontestingcomponentsinisolation.It often requires specific support, such as test harnesses or unit test frameworks. Component testing is normally performed by developers in their development environments.
- **Componentintegrationtesting** (alsoknownasunitintegration testing)focusesontestingthe interfaces and interactions between components. Component integration testing is heavily dependent on the integration strategy approaches like bottom-up, top-down or big-bang.

- **Systemtesting**focuses ontheoverallbehaviorandcapabilitiesofan entiresystem orproduct, often including functional testing of end-to-end tasks and the non-functional testing of quality characteristics.Forsomenon-functional qualitycharacteristics,itispreferable totestthemon a completesystemin arepresentative testenvironment (e.g.,usability). Usingsimulationsofsub-

systemsisalso possible.Systemtestingmay beperformedbyan independenttest team,andis related to specifications for the system.

- **Systemintegrationtesting** focuses ontestingtheinterfacesofthesystemundertest andother systems and external services . System integration testing requires suitable test environments preferably similar to the operational environment.
- **Acceptance testing** focuses on validation and on demonstrating readiness for deployment, whichmeans thatthesystemfulfillstheuser'sbusiness needs.Ideally,acceptancetestingshould be performed by the intended users. The main forms of acceptance testing are: user acceptance testing (UAT), operational acceptance testing, contractual and regulatory acceptance testing, alpha testing and beta testing.

Testlevels aredistinguishedbythefollowingnon-exhaustivelistofattributes, toavoidoverlappingoftest activities:

- Testobject
- Testobjectives
- Testbasis
- Defectsandfailures
- Approachandresponsibilities

2.2.2. TestTypes

Alotoftesttypesexist andcanbe appliedinprojects.In thissyllabus,thefollowing four testtypesare addressed:

Functionaltesting evaluatesthefunctionsthatacomponentorsystemshouldperform.Thefunctions are"what"thetest object should do. Themain objective offunctional testing is checking thefunctional completeness, functional correctness and functional appropriateness.

Non-functional testing evaluates attributes other than functional characteristics of a component or system.Non-functionaltestingisthetesting of"howwellthesystembehaves". The mainobjectiveof non-functional testing is checking the non-functional software quality characteristics. The ISO/IEC 25010 standard provides the following classification of the non-functional software quality characteristics:

- Performanceefficiency
- Compatibility
- Usability
- Reliability
- Security
- Maintainability
- Portability

Itissometimesappropriatefornon-functionaltesting tostartearlyinthe lifecycle(e.g., aspartof reviews and component testing or system testing). Many non-functional tests are derived from functional tests as

they use the same functional tests, but check that while performing the function, a non-functional constraint issatisfied (e.g.,checkingthat afunctionperformswithinaspecifiedtime,orafunctioncanbe ported to a new platform). The late discovery of non-functional defects can pose a serious threat to the successofa project.Non-functionaltestingsometimesneedsaveryspecifictestenvironment,suchasa usability lab for usability testing.

Black-boxtesting(seesection4.2)isspecification-basedandderivestestsfromdocumentationexternal to the test object. The main objective of black-box testing is checking the system's behavior against its specifications.

White-box testing (see section 4.3) is structure-based and derives tests from the system's implementation or internal structure (e.g., code, architecture, work-flows, and data flows). The main objectiveofwhite-boxtestingistocovertheunderlying structure bythetesttothe acceptablelevel.

All the four above mentioned test types can be applied to all test levels, although the focus will be differentateachlevel.Differenttesttechniquescan beusedtoderivetestconditionsandtestcases for all the mentioned test types.

2.2.3. ConfirmationTestingandRegressionTesting

Changesaretypicallymadetoa component orsystemto eitherenhanceitbyadding anewfeatureorto fix it by removing a defect. Testing should then also include confirmation testing and regression testing.

Confirmationtestingconfirms that anoriginaldefecthas beensuccessfullyfixed.Depending onthe risk, one can test the fixed version of the software in several ways, including:

- executingalltestcasesthat previouslyhavefailedduetothedefect,or,also by
- addingnewteststocoveranychangesthatwereneededtofixthedefect

However, when time or money is short when fixing defects, confirmation testing might be restricted to simplyexercisingthestepsthatshouldreproducethefailurecausedbythedefectandcheckingthatthe failure does not occur.

Regressiontestingconfirms thatnoadverseconsequenceshavebeencausedbyachange, includinga fix that has already been confirmation tested. These adverse consequences could affect the same component where the change was made, other components in the same system, or even other connected systems. Regression testing may not be restricted to the test object itself but can also be related tothe environment. Itis advisable first to perform animpact analysis tooptimizethe extent ofthe regression testing. Impact analysis shows which parts of the software could be affected.

Regression test suites are run many times and generally the number of regression test cases will increase with each iteration or release, so regression testing is a strong candidate for automation. Automation of these tests should start early in the project. Where CI is used, such as in DevOps (see section2.1.4),itis good practicetoalsoinclude automatedregressiontests.Depending onthesituation, this may include regression tests on different levels.

Confirmationtestingand/orregressiontestingforthetest objectareneeded onalltestlevelsifdefects are fixed and/or changes are made on these test levels.

2.3. MaintenanceTesting

There are different categories of maintenance, it can be corrective, adaptive to changes in the environmentorimproveperformanceormaintainability (seeISO/IEC14764fordetails),somaintenance can involve planned releases/deployments and unplanned releases/deployments (hot fixes). Impact analysismay bedonebefore achangeis made, tohelpdecideifthe changeshouldbe made, basedon thepotential consequencesin otherareasofthesystem. Testingthechangestoasysteminproduction includes bothevaluatingthe success of theimplementationofthe change andthechecking forpossible regressions in parts of the system that remain unchanged (which is usually most of the system).

Thescopeofmaintenancetestingtypicallydependson:

- Thedegreeofriskofthechange
- Thesize ofthe existingsystem
- Thesizeofthechange

Thetriggersformaintenance andmaintenancetestingcanbeclassifiedasfollows:

- Modifications,suchasplannedenhancements(i.e.,release-based),correctivechangesorhot fixes.
- Upgrades or migrations of the operational environment, such as from one platform to another, whichcanrequire testsassociated withthe newenvironmentaswellas ofthechangedsoftware, or tests of data conversion when data from another application is migrated into the system being maintained.
- Retirement,such aswhen anapplication reachestheend ofitslife.Whenasystemisretired, this canrequiretestingofdataarchivingiflongdata-retention periods are required. Testing ofrestore and retrieval procedures after archiving may also be needed in the event that certain data is required during the archiving period.