Sigurimi i Cilësisë Softuerike

Testimi i softuerit

Çfarë është testimi i softuerit dhe cilat janë objektivat e tij?

*"Testimi Softuerik është një proces **formal** i kryer nga **një ekip i specializuar i testimit** në të cilën një apo disa njësi softuerike të integruara ose një paketë e tërë softuerike janë ekzaminuar apo shqyrtuar duke **drejtuar (ang. run) programet në një kompjuter**. Të gjitha testimet e ngjajshme janë kryer në bazë të **procedurave të testimit të miratuar (aprovuar**) në **rastin e miratuar (aprovuar) të testimit.** "*

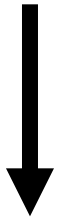
(Daniel Galin – Software Quality Assurance)

Çfarë është testimi i softuerit dhe cilat janë objektivat e tij?(vazhd.)

- Objektivat:
  - Gjen gabime në softuer
  - Rrit cilësinë e softuerit
  - Ekzekuton testimin brenda resurseve buxhetore të alokuara.

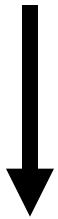
Ku përshtatet Planifikimi i Testimit me Testimin e Përgjithshëm

Përcaktimi i Metodologjisë së Testimit



Startegjitë e testimit: Big bang (zhurmë e madhe) ose strategjia rritëse e testimit (incremental testing strategy).

Nëse është në rritje, duhet të përdoret nga lart posht ose nga posht lart.



Test Planning

Çfarë është Planifikimi i Testimit?

- Planifikimi i Testimit është detyrë që duhet të ndërmerret para testimit softuerik.

- Planifikimi i Testimit mund të ndahet në këto nënpjesë:
  - **Çfarë** duhet të Testohet?
  - Cilat **raste të testimit** (**test cases**) do të përdoren?
  - **Kush** do të kryejë testimin?
  - **Mjedisi** i Testimit
  - Kur testi **përfundon**?

Çfarë duhet të Testohet?

- Natyra e testimit duhet të përcaktohet, kjo është për shkak se projekte të ndryshme kërkojnë qasje të ndryshme.

- Ka disa lloje të testimit që mund të kryhen, këto përfshijnë:
  - Unit Test
  - Integration Test
  - System Test

Çfarë duhet të Testohet? – Unit Test

- ***Unit Test***

Ky lloj apo tip i Planifikimit të Testimit shikon vetëm në komponentin e vogël apo module të cilat përbëjnë sistemin ose softuerin.

- *MSDN përcakton Unit Testimin si:*

*"Qëllimi kryesor i njësisë së testimit është që të marrë pjesën më të vogël të softuerit të testueshëm në aplikacion, izolon atë nga pjesa tjetër e kodit, dhe përcakton nëse ajo sillet tamam siç ne presim. Çdo njësi është testuar në veçanti para integrimit të tyre në module për të testuar ndërfaqet (*interfaces *) mes moduleve. Testimi në njësi (*Unit testing*) ka provuar vlerën e saj dhe një përqindje e madhe e defekteve janë identifikuar gjatë përdorimit të tij. "*

(http://msdn.microsoft.com/en-us/library/aa292197(VS.71).aspx)

Çfarë duhet të Testohet? – **Integration & System Test**

- ***Testimi i Integrimit***

Testimi i Integrimit përfshin testimin e disa moduleve apo komponentëve së bashku.

- ***MSDN definon integrimin e testimit si:***

"Testimi liIntegrimit është një zgjerim logjik i njësisë së testimit (unit testing). Në formën e tij më të thjeshtë, dy njësi që tashmë që janë testuar kombinohen në një komponentë dhe interfejsi mes tyre testohet "

(http://msdn.microsoft.com/en-us/library/aa292128.aspx)

- ***Testimi i Sistemit***

Testimi i Sistemit përfshin testimin e tërë të sistemit / softuerit.

Cilat burime janë përdorur për rastet e testimit?

- ***Çfarë është një rast testimi?***
  - Skenari i testimit të rastit me të cilin softueri performon dhe rezultatet regjistrohen.

Dy llojet e burimeve:

- ***Real Life test case***
  - Real life test rasti përfshin testimin e sistemit/softuerit me kushtet e botës reale.
  - Mund të identifikojë gabimet e papritura.

- ***Synthetic test case.***
  - Ky është një grup i skenarëve të përdorura për testim.
  - I mirë për diktim të gabimeve në dizajn.
  - I keq për diktim të gabimeve të papritura.

Kush do të kryejë testimin?

Unit Test

Internal Dev./Test Team

Integration Test

Internal Dev./Test Team

System Test

Internal Dev./Test Team

External Outsourced Vendors

Kush do të kryejë testimin? (Vazhd.)

- *Evidence of Research:* Software Testing Vendors:

- **Acutest** http://www.acutest.co.uk/
  

  - http://www.acutest.co.uk/acutest/clients

- **Thinksoftglobal** http://www.thinksoftglobal.com
  

  - http://www.thinksoftglobal.com/our_clients/client_listing/

- **Maverick independent testing** http://www.maveric-systems.com
  

  - http://www.maveric-systems.com/clients.php

- **e-testing** http://www.etesting.com
  - Goldman Sachs, O2, Sony Ericsson, Accenture, Halifax, Northern Rock, KMPG
  - http://www.etesting.com/clients_clients.html

Mjedisi i testimit

Developers Site

Target Site

Consultant Site

Kur do të përfundojë testimi?

Ka 5 metoda:

- ***Implementimi i rrugës së kompletuar***

Kjo rrugë përpiqet që të arrijë përsosmërinë duke plotësuar planin e testimit të përsëritur derisa nuk ka më gabime.

- ***Modelet matematikore të rrugës së aplikuar***

Kjo rrugë konsideron një model të zgjedhur aritmetik, ku vlerësimi bëhet për të parë gabimet që nuk janë zbuluar, bazuar në normën e gabimit.

- ***Gabimi i rrugës nga fillimi***

Kjo rrugë punon në parimin se numri i gabimeve të pazbuluara lidhet me numrin e gabimeve të zbuluara. Kur numri i gabimeve të pazbuluara arrin një nivel të miratimit, atëherë testimi do të përfundojë.

- ***Rruga e grupeve të pavarura të testimit***

Në këtë rrugë ka dy ekipe të testimit të cilat do të ndërmarrin reagime të testimit të sistemit / komponentit. Duke bërë kështu, prodhohen rezultate dhe referenca cross apo kryq nga rezultatet e dy ekipeve, numri i gabimeve të pazbuluara do të bjerë në mënyrë të konsiderueshme.

- ***Termination after resources have petered out. Përfundimi pasi burimet kanë përfunduar jashtë.***

Kjo rrugë thjesht përfundon testimin pasi burimet kanë dalë jashtë. Të tilla si koha, paraja, stafi, etj.

Software Testing and Quality Assurance

Unit Testing

Koncepti për njësinë e tesimitn – Unit Testing

- Static Unit Testing (Njësia e testimit statik)
  - Kodi është shqyrtuar mbi të gjitha sjelljet e mundshme që mund të lindin gjatë kohës së drejtuar
  - Kodi i çdo njësie është e vlefshme kundrejt kërkesave të njësisë duke shqyrtuar kodin
- Dynamic Unit Testing (Njësia e testimit dinamik)
  - Një njësi programi është ekzekutuar në të vërtetë dhe rezultatet e saj janë vërejtur
  - Vëzhgohen disa sjellje tipike të programit, dhe arrihet në një përfundim në lidhje me cilësinë e sistemit
- Testimi i njësisë statike nuk është një alternativë për të testimin e njësisë dinamike
- Analiza statike dhe dinamike janë komplementare apo plotësues në natyrë.
- Në praktikë, testimi i pjesshëm i njësisë dinamike kryhet njëkohësisht me testimin e njësi statike.
- Është e rekomanduar që testimi i njësisë statike të kryhet para fillimit të testimit të njësisë dinamike

Testimi i njësisë statike
Static Unit Testing

- Në kodin e testimit të njësisë statike është rishikuar duke aplikuar teknika:
  - **Inspektim:** Është hap pas hapi shqyrtim i grupit të nxjerrë të një produkti të punës, me çdo hap të kontrolluar kundrejt kritereve të para-përcaktuara
  - **Ecje-përmes - Walkthrough:** Është shqyrtim ku autori udhëheq ekipin përmes një manuali ose të simuluar për ekzekutim të produktit duke përdorur skenare të para-përcaktuar
- Ideja këtu është që të shqyrtojë kodin burimor në detaje në mënyrë sistematike
- Objektivi i rishikimit të kodit është të shqyrtojë kodin, dhe jo të vlerësojë autorin e kodit
- Rishikimi kodi duhet të planifikohet dhe menaxhohet në mënyrë profesionale
- Çelësi i suksesit të kodit është për të ndarë dhe të pushtuar apo sunduar
  - Ekzaminuesi inspekton pjesë të vogla të njësisë në izolim
    - asgjë nuk është anashkaluar
    - korrektësia e të gjitha pjesëve të shqyrtuara të modulit nënkupton saktësinë e të gjithë modulit

Static Unit Testing (Code Review)

- Hapi 1: **Readiness - Gatishmëri**
  - **Criteria - Kriteret**
    - **Completeness - Kompletimi**
    - **Minimal functionality – Funksionaliteti minimal**
    - **Readability - Lexueshmëri**
    - **Complexity - Kompleksiteti**
    - **Requirements and design documents - Kërkesat dhe dokumentet e projektimit**
  - **Roles - Rolet**
    - **Moderator**
    - **Author**
    - **Presenter**
    - **Record keeper**
    - **Reviewers**
    - **Observer**
- Hapi 2: **Preparation - Përgatitja**
  - **List of questions - Lista e pyetjeve**
  - **Potential Change Request (CR) – Kërkesa me ndryshime potenciale**
  - **Suggested improvement opportunities - Mundësitë për përmirësime të sugjeruara**

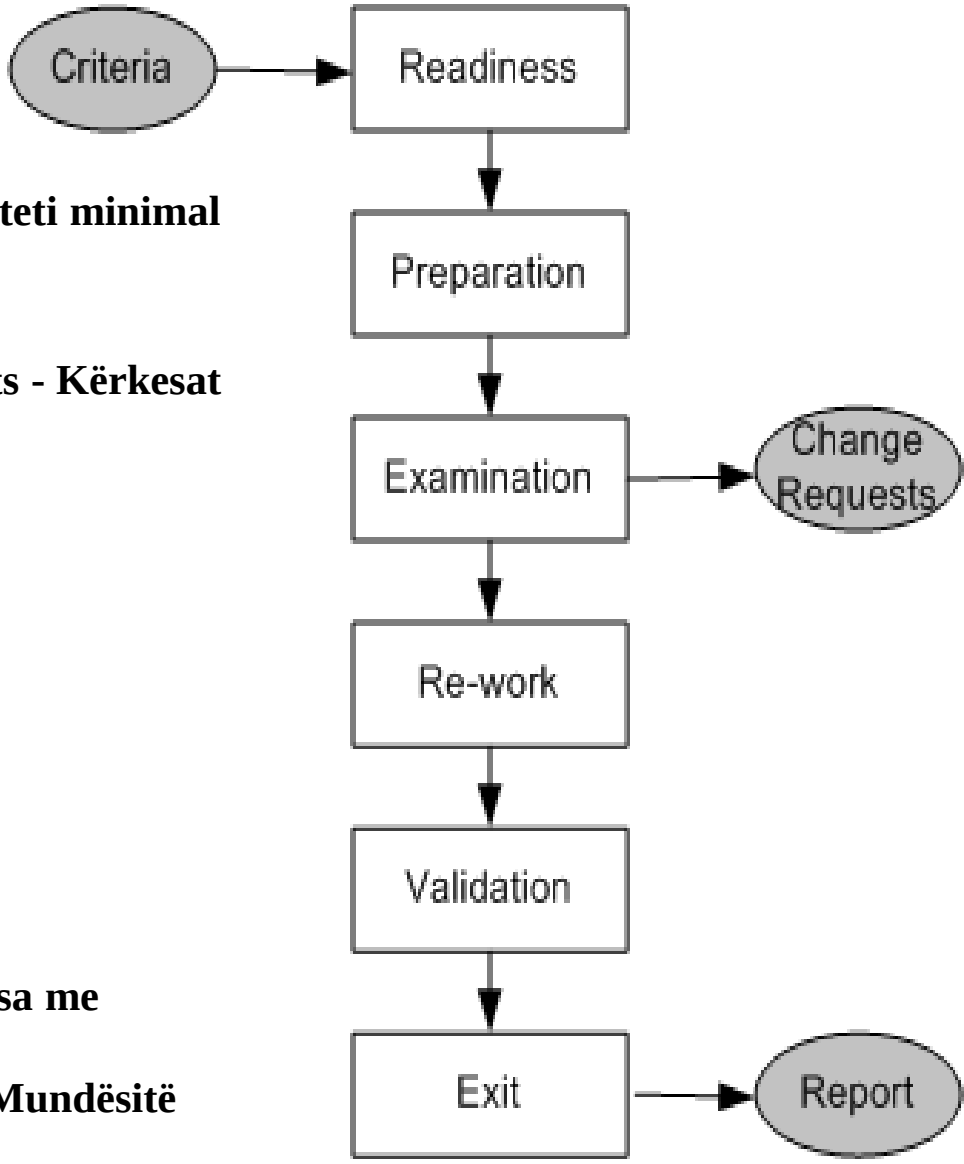


Figure 3.1: Steps in the code review process

Static Unit Testing (Code Review)

- Hapi 3: **Examination - Ekzaminimi**
  - Autori bën një prezantim
  - Prezantuesi lexon kodin
  - Personi që mban shënime dokumenton CR (kërkesat e ndryshuara)
  - Moderatori siguron që shqyrtim të jetë në rrugë të mbarë
- Hapi 4: **Re-work**
  - Bëni listën e të gjitha CRs
  - Bën një listë të përmirësimeve
  - Regjistron minutat e takimit
  - Autori punon në CR për të rregulluar këtë çështje
- Hapi 5: **Validation - Validimi**
  - CR-të janë të validuara në mënyrë të pavarur
- Hapi 6: **Exit - Dalja**
  - Një raport përmbledhës i mbledhjes shpërndahet

Change Request (CR) Ndryshimi i kërkesës përfshin detajet e mëposhtme:

- Jepet një përshkrim i shkurtër i çështjes
- Caktohet një nivel i prioritetit (të mëdha ose të vogla) në një **CR**
- Caktohet një person për të ndjekur atë
- Vendoset një afat kohor për adresimin e një **CR**

Static Unit Testing (Code Review)

Metrikat e mëposhtme mund të mblidhen nga një rishikim i kodit:

- Numri i linjave të kodit (LOC) i rishikuar në orë
- Numri i CR të krijuara për një mijë rreshta të kodit (KLOC)
- Numri i CR i gjeneruar në orë
- Numri i përgjithshëm i orëve të shpenzuar në procesin e rishikimit të kodit

Static Unit Testing (Code Review)

- Metodologjia e rishikimit të kodi mund të jetë i zbatueshëm për të shqyrtuar dokumente të tjera.
- Pesë lloje të ndryshme të dokumenteve të sistemit janë të krijuar nga departamenti i inxhinierisë:
  - Kërkesa
  - Specifikimet funksionale
  - Dizajnimi apo projektimi i nivelit të lartë
  - Dizajnimi apo projektimi i nivelit të ulët
  - Kodi
- Në instalimin shtesë, përdoruesi, dhe probleme e udhëzimeve janë zhvilluar nga grupi i dokumentacionit teknik

Hierarchy of System Documents
Requirement: High-level marketing or product proposal.
Functional Specification: Software Engineering response to the marketing proposal.
High-Level Design: Overall system architecture.
Low-Level Design: Detailed specification of the modules within the architecture.
Programming: Coding of the modules.

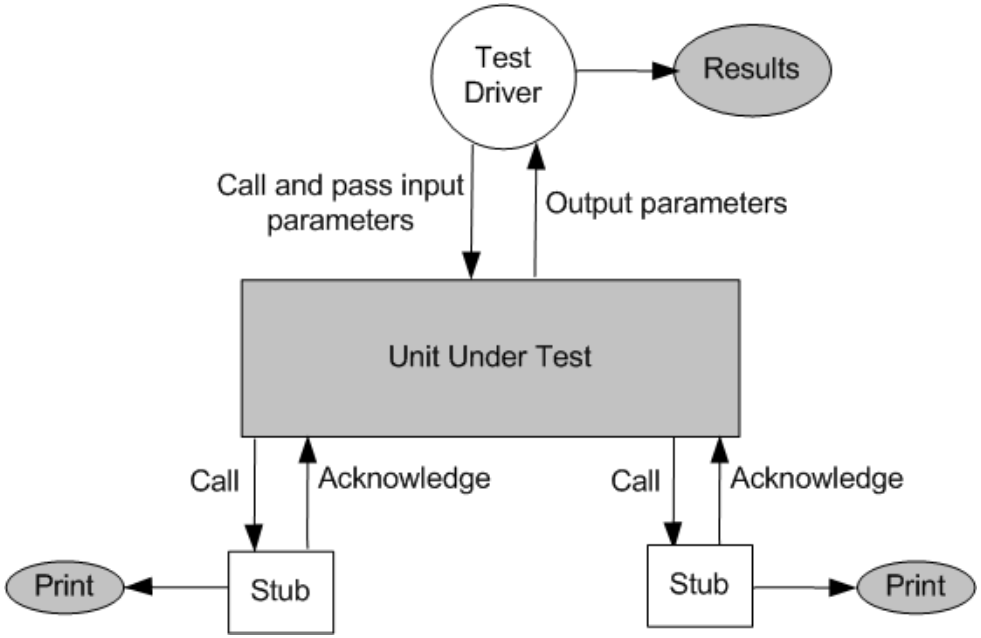
Parandalimi i defekteve apo të metave

- Ndërton kodin në kod
- Përdor kontrollin standard për të zbuluar dukuritë e mundshme të kushteve të gabimit
- Siguron që kodi të ekzistojë për të gjitha vlerat e kthimit apo të numëruara
- Siguron që fushat e të dhënave kthyese apo të numëruara dhe të buffer-it të trajtohen në mënyrë të përshtatshme
- Siguron mesazhet e gabimeve dhe ndihmon tekstet nga një burim i përbashkët
- Validon të dhënat hyrëse
- Përdor pohimet për të zbuluar kushtet e pamundura
- Len pohimet në kod.
- Plotësisht dokumenton pohimet që duken të paqarta.
- Pas çdo llogaritje të madhe të kundërt – llogarit hyrjet nga rezultatet në kodin e vetë
- Përfshin një loop counter brenda secilës loop

Njësia e testimit dinamik - Dynamic Unit Testing

- Mjedisi i një njësie është rivalizuar (emulated) dhe testuar në izolim
- Njësia e thirrësit është i njohur si *shofer testimi (test driver)*
  - Shofer testimi është një program që thërret njësinë e nën testimit - unit under test (UUT)
  - Ajo ofron të dhëna hyrëse për njësinë e testimit dhe të raporton rezultatet e testimit
- Emulacioni i njësive i thirrur nga UUT quhen *stubs*
  - Është një program dummy
- *Test driver* dhe *stubs* së bashku quhen *scaffolding*
- Dokumentimi i dizajnit në nivel të ulët jep udhëzime për zgjedhjen e të dhënave hyrëse të testimit

Mjedisi i njësisë të testimit dinamik



Njësia e testimit dinamik - Dynamic Unit Testing

Përzgjedhja e të dhënave të testimit në përgjithësi bazohet mbi teknikat e mëposhtme:

- Control flow testimi
  - Vizaton grafik për kontrollin e rrjedhjes (CFG - control flow graph) nga një njësi e programit
  - Selekton kritere nga disa testime të kontrollit të rrjedhjes
  - Identifikimi i një rrugë në CFG për të kënaqur kriteret e përzgjedhjes
  - Nxjerr rrugën e shpreshjes së predikatit nga rrugët e selektuara
  - Duke zgjedhur shprehjen e predikatit për një rrugë, mund të gjenerohen të dhëna
- Data flow testimi
  - Vizaton grafik të rrjedhjes së të dhënave (DFG- data flow graph) nga një njësi e programit dhe pastaj ndjek procedurën e përshkruar në testimin e rrjedhjes së kontrollit.
- Domain testimi
  - Gabimet e domenit janë definuar dhe pastaj të dhënat e testimit janë zgjedhur për të kapur ato gabime
- Functional program testimi
  - Domenet hyrëse/dalëse janë definuar për të llogaritur vlerat e të dhënave hyrëse të cilat shkaktojnë që njësia të prodhojë vlerat e pritura dalëse apo të prodhuara

Ndryshimet e testimit - Mutation Testing

- Modifikon një program duke futur një ndryshim të vetëm të vogël në kod.
- Një program i modifikuar quhet *mutant*.
- Një mutant *vritet* kur ekzekutimi i çështjes së testimit dështon. Mutanti në këtë rast konsiderohet të jetë i *vdekur.*
- Një mutant është *ekuivalent* me programin e dhënë nëse ajo gjithmonë prodhon të njëjtën dalje si program origjinal.
- Një mutant quhet i aftë për të vrarë ose kokëfortë, nëse grupi ekzistues i rasteve të testimit është i pamjaftueshëm për të vrarë atë.
- Një *rezultat* i mutacionit për një grup të rasteve të testimit është përqindja e mutantëve jo-ekuivalente të *vrarë* nga testimi i përcjellur.
- Përcjellja e testimit është *mutacion-adekuat* në qoftë se rezultati i tij i mutacionit është 100%

Mutation Testing

Testimi i mutacionit bën dy supozime kryesore:

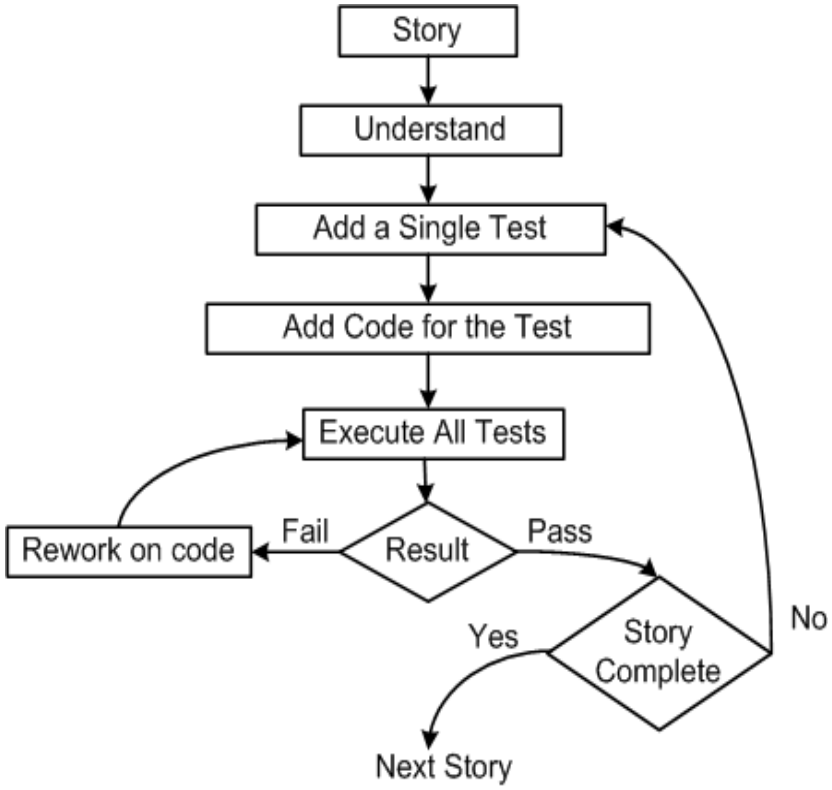
- Hipoteza e progrmauesit kompetent
  - Programuesit janë përgjithësisht kompetent dhe ata nuk krijojnë programe të rastit
- Efektet për bashkim
  - Gabimet komplekse janë shoqëruar me gabimet e thjeshta në mënyrë të tillë që një suitë apo përcjellje e testimit zbulon gabimet e thjeshta në një program që do të zbulojë shumicën e gabimeve komplekse.

Debugging

- Procesi i përcaktimit të shkakut të dështimit është i njohur si *debugging apo debagim.*
- Konsumon kohë dhe procese të gabimeve.
- Debagimi përfshin një kombinim të vlerësimit sistematik, intuitës dhe pak fat
- Qëllimi është të izolohet dhe përcaktohet shkaku i saj i veçantë, duke pasur parasysh simptomën e problemit
- Ka tre mënyra për debagim
  - Brute force (me forcë)
  - Cause elimination (eleminimi i shkakut)
    - Induction - induksioni
    - Deduction - deduksioni
  - Backtracking (kthimi prapa)

Unit Testing in eXtreme Programming

1. Pick a requirement, i.e., a story
    1. Zgjidh një kërkesë, dmth, një histori
2. Write a test case that will verify a small part of the story and assign a fail verdict to it
    1. Shkruaj një rast testimi që do të verifikojë një pjesë të vogël të historisë dhe cakton një vendim që do të dështojë.
3. Write the code that implement particular part of the story to pass the test
    1. Shkruaj kodin qe zbaton një pjesë të veçantë të historisë që të kalojnë testin
4. Execute all test
    1. Ekzekuto gjithë testimin
5. Rework on the code, and test the code until all tests pass
    1. Puno në kod, dhe testo kodin derisa i gjithë testimet të kalohen me sukses
6. Repeat step 2 to step 5 until the story is fully implemented
    1. Përsërit hapin 2 dhe hapin 5 deri sa historia është implementuar plotësisht



Test-first process in XP

JUnit – A Framework for Unit Testing

- JUnit: It is a framework for performing unit testing of Java programs.
  - Other frameworks: NUnit (C#), CPPUnit (C++), fUnit (Fortran)
- Intuitive steps to test a method in Java (Ex. Move() method of PlanetClass)
  - **Create** an object instance of PlanetClass. Call it Mars.
  - **Select** values of all input parameters of Move().
  - **Compute** the expected value to be returned by Move(). Let it be y.
  - **Execute** method Move() on Mars with the selected input values.
    - Let Move() return a value called z.
  - **Compare** the actual output (z) returned by Move() with the expected value (y).
    - If (z == y), Move() ***passes*** the test; otherwise it ***fails***. ← **Report** the result.
- JUnit makes writing of test cases easier. ➔ Next slide …

JUnit – A Framework for Unit Testing

- JUnit provides a basic class called TestCase.
- The tester
  - *Extends* the TestCase class for each test case. 10 extensions for 10 test cases.
  - Alternatively, extend TestCase to have 10 methods for 10 test cases.
- The TestCase class provides methods to make *assertions*.
  - assertTrue(Boolean condition)
  - assertFalse(Boolean condition)
  - assertEquals(Object expected, Object actual)
  - assertEquals(int expected, int actual)
  - assertEquals(double expected, double actual, double tolerance)
  - assertSame(Object expected, Object actual)
  - assertNull(Object testobject)
  - …
- The tester can have her own assertions.

JUnit – A Framework for Unit Testing

- Each assertion accepts an optional *first* parameter of type String; if the assertion **fails**, the string is displayed. ← Help for the tester…
- The assertEquals() method displays a message upon failure.
    - junit.framework.AssertionFailedError: expected: <x> but was: <y>
- Note that only failed tests are reported.
- The following shows how assertTrue() works.

```
static public void assertTrue(Boolean condition) {
        if (!condition)
            throw new AssertionFailedError();
}
```

Figure 3.5: The assertTrue() assertion throws an exception

JUnit – A Framework for Unit Testing

```
import TestMe; // TestMe is the class whose methods are going to be tested.
import junit.framework.*; // This contains the TestCase class.

public class MyTestSuite  extends  TestCase { // Create a subclass of TestCase

    public void MyTest1() { // This method is the first test case
        TestMe object1 = new TestMe( ... ); // Create an instance of TestMe  with desired
     params
        int x = object1.Method1(...); //  invoke Method1 on object1
        assertEquals(365, x); // 365 and x are expected and actual values, respectively.
    }

    public void MyTest2() { // This method is the second test case
        TestMe object2 = new TestMe( ... ); // Create another instance of
                                  // TestMe with desired parameters
        double y = object2.Method2(...); //  invoke Method2 on object2
        assertEquals(2.99, y, 0.0001d); // 2.99 is the expected value;
                                  // y is the actual value;
                                  // 0.0001 is tolerance level
    }
}
```

Figure 3.5: An example test suite

Tools For Unit Testing – Mjetet për njësinë e testimit

- Auditimi i kodit (Code auditor)
  - Ky mjet është përdorur për të kontrolluar cilësinë e softuerir për të siguruar që ajo i plotëson disa standarde minimale për kodim
- Kontrollimi i detyruar (Bound checker)
  - Ky mjet mund të kontrollojë shkrimet aksidentale në zonat e udhëzimit të memorjes, ose në vende tjera të memorjes jashtë zonës së magazinimit të të dhënave të aplikimit.
- Dokumentimi (Documenters)
  - Këto mjete lexojnë burimin e kodit dhe automatikisht gjeneron përshkrimet dhe telefonuesi ose modeli i të dhënave nga burimi i kodit
- Debagimi interaktiv (Interactive debuggers)
  - Këto mjete të ndihmojnë zhvilluesit e programeve në zbatimin e teknikave të ndryshme të debagimit

    Shembull: Breakpoint dhe Omniscient debuggers
- Emulatorët në qark (In-circuit emulators)
  - Siguron një lidhje me shpejtësi të lartë të Ethernet në mes host debugger (debagimit të hostuar) dhe mikroprocesorit të synuar, duke u mundësuar zhvilluesve të kryejnë nivel të burimit të kodit për debagim.

Tools for Unit Testing – Mjetet për njësinë e testimit

- Memory leak detectors – Detektorët e kujtesës rrjedhëse
  - Këto mjete testojnë shpërndarjen apo alokimin e kujtesës apo memorjes e cila dërgon kërkesë në memorje dhe dështon për mos alokim në memorje
- Static code (path) analyzer – Analiza e kodit statik (rrugës)
  - Këto mjete identifikojnë rrugët për testim të bazuar në strukturën e kodit si psh. McCabe kompleksiteti

Tools for Unit Testing – Mjetet për njësinë e testimit

- Mbështetja e inspektimit softuerik
  - Mjetet mund të ndihmojnë në orarin për inspektim të grupit
- Analizatorët e testimit të mbulimit
  - Këto mjete masin mbulim e brendshëm të testimit, shpesh të shprehura në aspektin e strukturës së kontrollit të objektit të testimit, dhe raporton mbulimin e metrikës.
- Gjeneratori i testimit të të dhënave
  - Këto mjete ndihmonjë programuesit në zgjedhjen/selektimin e të dhënave të tëstimit të cilat shkaktojnë që programi të sillet në një mënyrë të dëshiruar
- Testimi harness
  - Kjo klasë e mjeteve mbështet ekzekutimin e testimit dinamik
- Monitorimi i performancës
  - Karakteristikat e kohës të komponentëve të softuerit monitorohet dhe vlerësohet nga këto mjete
- Analizatorët e rrjetit
  - Këto mjete kanë aftësinë për të analizuar trafikun dhe identifikuar fushat me probleme
  - These tools have the ability to analyze the traffic and identify problem areas

Tools for Unit Testing – Mjetet për njësinë e testimit

- Simulatorët dhe emulatorët
  - Këto mjete janë përdorur për të zëvendësuar softuerin dhe hardverin e vërtetë që nuk janë aktualisht në dispozicion. Të dy llojet e mjeteve janë përdorur për trajnime, siguri, dhe qëllime ekonomike
- Gjeneratorët e trafikut
  - Këto prodhojnë rrjedha të transaksioneve apo pako të të dhënave.
- Versioni i kontrollit
  - Një version i sistemit të kontrollit ofron funksionalitete për të ruajtur një sekuencë të rishikimeve të programeve dhe fajlle me informacione të lidhura në zhvillim e sipër