# УНИВЕРЗИТЕТ У БЕОГРАДУ

**ФАКУЛТЕТ ОРГАНИЗАЦИОНИХ НАУКА**

**Опис и примена алата за тестирање софтверског система на свим нивоима архитектуре софтверског система**

Ментор Студент

Др Саша Лазаревић Катарина Симић 2020/3701

Београд, 2021. године

Cадржај

[Тестирање софтвера 3](#_Toc66892544)

[Алати за тестирање презентационог слоја 3](#_Toc66892545)

[Алати за тестирање аликативног слоја 3](#_Toc66892546)

[Алати за тестирање слоја података 3](#_Toc66892547)

[Литература 4](#_Toc66892548)

# 

# Тестирање софтвера

Тестирање софтвера пружa објективан поглед на софтвер како би се омогућило предузећу да сагледа и разуме ризик софтверске имплементације. Технике тестирања укључују процес извршавања програма или апликације са намером проналажења кварова и провере да је софтверски производ погодан за употребу.[2]

Свеобухватно тестирање укључује тестирање софтверског система кроз сва три нивоа архитектуре, и то на нивоу презентационог, апликативног слоја у оквиру кога се може наћи слој приступа подацима и на самом крају - слој података. У наставку ће бити описани алати помоћу којих је могуће тестирати наведене слојеве софтверске архитектуре.

# Алати за тестирање презентационог слоја

Презентациони слој треба тестирати до одређене мере, као и све друге слојеве у апликацији. Потребно је тестирати како кориснички интефејс реагује када корисник изврши одређену акцију, нпр. подаци би морали да имају коректан ток ка корисничком интерфејсу и из њега. Тестирање корисничког интерфејса, често названо *UI* (енг. *User Interface*) тестирање спада у ред аутоматизованих тестова на високом нивоу апстракције [8].

Данас постоји низ квалитетних и напредних алата за *UI* тестирање. Развој ових алата полако мења парадигму, ширину и могућности тестирања на овом нивоу. Неки од тих алата се користе независно од платформе, техничким услова, програмским језицима итд. Постоји низ специјализираних алата, од којих се истичу Selenium и Аpache ЈМетер као најпопуларнији оквири за аутоматскох тестирање [8].

Оно што је заједничко за већину ових алата је да не захтевају никакво програмерско знање. Основни принцип њиховог функционисања се може поделити у 5 корака [8]:

1. Покренути снимање теста
2. Извршити операције потребне за тест над интерфејсом система за тестирање
3. Дефинисати жељени исход
4. Прекинути снимање теста
5. Покренути снимљени тест

Један од најпознатијих алата за овакво тестирање је Селениум. Једна од верзија Селениум

алата подржава и тестирање веб апликација и може се инсталирати као додатак за одређени претраживач. За разлику од већине других алата, Селениум има могућност креирања скрипти у више различитих програмских језика, тако да програмери могу да их измене и допуне по потреби [8].

За Windows апликације, један од често коришћених алата је WinRunner. Овај алат подржава креирање мапе графичког корисничког интерфејса, која омогућава идентификовање елемената интерфејса по задатим критеријумима и праћење акција које се над објектима у мапи извршавају. Такође, могуће је снимити и тестове који прате кретање курсора по екрану, а који су погодни за тестирање апликација које имају функционалности цртања [8].

TestComplete је још један од алата који омогућавају тестирање апликација кроз њихов графички кориснички интерфејс. Он подржава и вишеструке валидације у току једног теста, што значи да можемо у току извршавања неког комплексног сценарија, више пута проверити стање система у различитим тренуцима. Ова могућност знатно олакшава проналажење грешке у систему када се неки од сложених тестова неуспешно изврши [8].

У наставку су наведени још неки популарнији алати [7]:

• Carina (линк https://github.com/qaprosoft/qps-infra)

• Google EarlGrey (линк https://github.com/google/EarlGrey)

• Cucumber (линк https://cucumber.io/)

• Watir (линк http://watir.com/)

• Appium (линк https://github.com/appium/appiumdesktop/releases/tag/v1.13.0)

• RobotFramework (линк https://robotframework.org)

• Apache JMeter (линк https://jmeter.apache.org/download\_jmeter.cgi)

• Gauge (линк https://gauge.org/getting-started-guide/we-start/)

• i Robotium (линк <https://github.com/RobotiumTech/robotium>)

## Примена Селениум алата у оквиру .НЕТ пројекта

Да би се овај алат користио потребно је креирати пројекат за тестирање и у оквиру њега додати путем NuGet пакет менаџера две библиотеке: *Selenium.WebDriver* и *Selenium.Chrome.WebDriver*. На слици је приказан почетак писања теста, тест имплементира интерфејс јер је неопходно затворити прозор прегледача једном када се тест заврши и такође, неопходно је ослободити задржане ресурсе.

[TestCase("mBroj","spanMBroj")]

[TestCase("ime", "spanIme")]

[TestCase("prezime", "spanPrezime")]

[TestCase("datumRodjenja", "spanDatumRodjenja")]

[TestCase("email", "spanEmail")]

[TestCase("mestoId", "spanMesto")]

public void CreateOsobaEmptyField(string elementId, string spanId)

{

\_driver.Navigate()

.GoToUrl("https://localhost:44341/Osoba/Create");

\_driver.FindElement(By.Id(elementId)).SendKeys("");

\_driver.FindElement(By.Id("btnCreate")).Click();

var errorMessage = \_driver.FindElement(By.Id(spanId)).Text;

Assert.AreEqual("Create - MAS\_TestiranjeSoftvera\_Projekat", \_driver.Title);

string expectedMessage = string.Empty;

switch(elementId)

{

case "mBroj":

expectedMessage = "Unesite maticni broj!"; break;

case "ime":

expectedMessage = "Unesite ime!"; break;

case "prezime":

expectedMessage = "Unesite prezime!"; break;

case "datumRodjenja":

expectedMessage = "Unesite datum rodjenja!"; break;

case "email":

expectedMessage = "Unesite email!"; break;

case "mestoId":

expectedMessage = "Odaberite mesto!"; break;

default: break;

}

Assert.AreEqual(expectedMessage, errorMessage);

}

Метода FindByElement проналази одговарајуће поље и методом SendKeys га попуњава, а затим кликом на дугме шаље захтев. Будући да се ради у случају када се у поља уносе празне вредности, очекује се да валидација одради своје и у одговарајући HTML елемент упише поруку о насталим грешкама.

# Алати за тестирање аликативног слоја

Тестирање се најчешће посматра као доказ да програми исправно раде, али заправо,

програми се тестирају да би се доказало постојање грешака. Пошто је циљ да се грешка открије, тест се сматра успешним само ако се грешка открије или ако дође до отказа у току тестирања [3].

*Test Driven Development* је техника развоја софтвера која подразумева често тестирање написаног кôда. Тест се пише пре кôда, а након тога се пише изворни кôд који треба да задовољи тест. Рефакторисањем се тај кôд даље пречишћава и поједностављује, али основно је да тест који се једном верификовао мора да се изнова верификује, при свим следећим изменама јединичним тестирањем (unit test). Постоје и функционални тестови кроз тест примере, односно тест случајеве које изводи сам клијент. Тестови корисности и корисничког интерфејса се такође изводе код клијента [5].

Данашњи софтверски системи су веома велике и комплексне апликације. Зато је важно користити алате за аутоматизовано извршавање тестова, јер омогућавају веома велики број случајева за тестирање, без којих систем би био површно тестиран. Алати за извршавање тестова могу да се интегришу са другим алатима ради изградње окружења за свеобухватно тестирање. Алати се често повезују са базом података за тестирање, мерним алатима,

алатима за анализу кода, едиторима текста и алатима за симулацију и моделовање да би се аутоматизовао што је могуће већи део процеса тестирања. Међутим, утврђивање да грешка постоји не значи да је грешка и пронађена. Зато ће аутоматизовано тестирање захтевати и људски труд у тражењу извора за настали проблем [5].

JUnit је open source оквир за тестирање, написан за Јава програмски језик. Међутим, постао је толико популаран да је преведен и прилагођен за коришћење у другим програмским језицима: C# (NUnit), Python (PyUnit), Fortran (fUnit), C++ (CPPUnit). Помоћу ових алата, могуће је аутоматизовано извршавати јединичне тестове из развојног окружења.

## Nunit алат

Први NUnit алати обезбеђивали су основне класе из којих су се развијале класе за тестирање. Међутим, ови системи имају доста ограничења приликом развоја кода за тестирање зато што велики број програмских језика (Јава и C#) омогућава само једноструко наслеђивање. Ово значи да је рефакторинг тест кода немогућ без упознавања компликованих хијерархија наслеђивања. Касније .NET уводи нови концепт програмирања који решава овај проблем: атрибути. Атрибути омогућавају додавање мета-података коду. Атрибути не утичу на извршавање самог кода, они се најчешће користе за документовање кода, али могуће је користити их како би додали информације о .NET асемблију [4].

Примена атрибута представља основу за рад NUnit 2.0 верзије. Апликација Test Runer скенира компајлирани код како би нашла аттрибуте који указују које су класе и методе креиране за тестирање, да би се затим рефлексијом извршиле те методе. Није потребно извлачити тест класе из основне класе, већ треба користити праве атрибуте. NUnit омогућава додавање великог броја атрибута који се користе приликом креирања тестова.

Постоје и атрибути који указују на очекиване грешке или који указују на тест који ће бити изостављен [4].

### Атрибут TextFixture

TextFixture се користи како би указао да класа садржи тест методу. Приликом извршавања

апликације за тестирање, апликација скенира код како би пронашла тест класе. Наредни код илуструје коришћење овог атрибута [1].

namespace UnitTestingExamples

{

using System;

using NUnit.Framework;

[TestFixture]

public class SomeTests

{

….

}

}

# 

# Класе које користе атрибут TestFixture морају имати public конструктор који нема улазних атрибута и који враћа void.

## Атрибут Test

Овај атрибут се користи како би указао да апликација која покреће тест треба да изврши методу која је означена овим параметром. Потребно је да ова метода буде јавна, нема повратну вредност и не захтева улазне параметре. Метода која не задовољава ове услове неће бити приказана у Test Runner GUI и неће бити извршена приликом покретања теста [1].

namespace UnitTestingExamples

{

using System;

using NUnit.Framework;

[TestFixture]

public class SomeTests

{

[Test]

public void TestOne()

{

// Do something...

}

}

}

### Атрибути SetUp & Teardown

Приликом састављања тестова, често се наилази на случајеве у којима је пре и након сваког теста потребно позвати велики број функција. Овим функцијама се најчешће креирају зависни објекти (нпр. конекција ка бази података). Једно решење овог проблема је креирање приватне методе која се позива из свих тест метода. Друго решење је коришћење атрибута Setup and Teardown. Ови атрибути говоре да ће се функција извршити пре (Setup) или након (Teardown) сваке тест методе у оквиру класе која је означена атрибутом Test Fixture [1].

namespace UnitTestingExamples

{

using System;

using NUnit.Framework;

[TestFixture]

public class SomeTests

{

private int \_someValue;

[SetUp]

public void Setup()

{

\_someValue = 5;

}

[TearDown]

public void TearDown()

{

\_someValue = 0;

}

[Test]

public void TestOne()

{

// Do something...

}

}

}

### Атрибут ExpectedException

Атрибут ExpectedException се користи када је потребно проверити да ли програм враћа грешку одређеног типа. Алтернатива коришћењу овог атрибута било би креирање try, catch блока у оквиру којег се поставља логичка вредност одређене промењиве [1].

namespace UnitTestingExamples

{

using System;

using NUnit.Framework;

[TestFixture]

public class SomeTests

{

[Test]

[ExpectedException(typeof(InvalidOperationException))]

public void TestOne()

{

// Do something that throws an InvalidOperationException

}

}

}

Тест ове методе ће бити успешан само ако програм враћа грешку типа InvalidOperationException. У ситуацијама када програм враћа више од једног типа грешке потребно је користити последњу грешку у стеку. Тест може тестирати само једну ствар. Друга специфицност је да атрибут нема информацију о наслеђивању. Ако програм враћа грешку чији тип наслећује класу InvalidOperationException, тест неће бити успешан [1].

### Атрибут Ignore

Овај атрибут користи се у случајевима када није потребно тестирати одређену тест методу и када је потребно привремено склонити тест [1].

namespace UnitTestingExamples

{

using System;

using NUnit.Framework;

[TestFixture]

public class SomeTests

{

[Test]

[Ignore("We're skipping this one for now.")]

public void TestOne()

{

// Do something...

}

}

}

### Класа NUnit Assertion

NUnit осим атрибутима који се користе како би идентификовали тестове у коду, има и класу Assertion. Ова класа садржи статичке методе које се могу користити за тестирање претпоставки односно упоређивање добијених и очекиваних резултата [1].

namespace UnitTestingExamples

{

using System;

using NUnit.Framework;

[TestFixture]

public class SomeTests

{

[Test]

public void TestOne()

{

int i = 4;

Assertion.AssertEquals( 4, i );

}

}

}

### Извршавање тестова

NUnit долази са две различите апликације за покретање тестова: Windows GUI апликацијом и конзолном XML апликацијом. За покретање GUI апликације неопходно је навести локацију на којој се налазе скуп унит тестова одређеног асемблија. Скуп тестова је библиотека класа или извршних програма која садржи атрибут Test Fixtures. Приликом покретања тестова у апликацији се може графички видети хијерархија тест класа и одговарајућих тест метода који припадају одговарајућој класи [4].

За покретање свих тестова потребно је одабрати опцију Run. Ако је потребно тестирати само методе одговарајуће класе Test Fixture или само једну тест методу, потребно је двапут кликнути на одговарајућу класу/методу [4].

Ако је потребно аутоматизовати процес тестирања тако да се добије излаз који се може поставити на веб сајт или неко друго место где могу приступити развојни тим, менаџмент или корисници, тада GUI апликација није најбоље решење. Конзолна апликација NUnit 2.0 узима локацију бибилотеке класа као аргумент командне линије и као резултат враћа XML. Одговарајућим XSLT трансформацијама могуће је конвертовати XML у HTML или неки други формат [4].

# Алати за тестирање слоја података

# Литература

1. Khorikov, V. (2020). Unit Testing Principles, Practices, and Patterns: Effective testing styles, patterns, and reliable automation for unit testing, mocking, and integration testing with examples in C# (1st ed.). Manning Publications. <http://sd.blackball.lv/library/Unit_Testing_(2020).pdf>
2. IEE (1990). *IEEE Standard Glossary of Software Engineering Terminology.* [На мрежи]. Доступно: [doi](https://en.wikipedia.org/wiki/Doi_(identifier))[:10.1109/IEEESTD.1990.101064](https://doi.org/10.1109%2FIEEESTD.1990.101064), [ISBN](https://en.wikipedia.org/wiki/ISBN_(identifier)) 9781559370677 [Последњи приступ 14.03.2021.]

[3] Osherove, R. (2009). The Art of Unit Testing: With Examples in .net (1st ed.). Manning Publications. <http://accorsi.net/docs/TheArtofUnitTesting.pdf>

[4] Hunt, A., Thomas, D., & Hargett, M. (2007). Pragmatic Unit Testing in C# with NUnit, 2nd Edition (Pragmatic Starter Kit Series, Vol. 2) (Second ed.). Pragmatic Bookshelf. <http://index-of.es/Programming/Pragmatic%20Programmers/Pragmatic%20Unit%20Testing%20In%20CSharp%20with%20NUnit%202nd%20Edition.pdf>

[5] Iii, G. M. (2013). Test-Driven Database Development: Unlocking Agility (Net Objectives Lean-Agile Series) (1st ed.). Addison-Wesley Professional.

[6] Database testing. (2016). [E-book]. <https://www.tutorialspoint.com/database_testing/database_testing_tutorial.pdf>

[7] Kazeeva, A. (2019, May 16). 10 Best Open Source Test Automation Frameworks for Every Purpose. Dzone.Com. <https://dzone.com/articles/10-best-open-source-test-automation-frameworks-for>

[8] McCaffrey, J. D. (2012). .NET Test Automation Recipes: A Problem-Solution Approach [E-book]. aPress. <http://sd.blackball.lv/library/NET_Test_Automation_Recipes.pdf>