Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Analýza využití moderních metod testování software v dostupných nástrojích

Plzeň 2016 Patrik Bezděk

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 3. května 2016

Patrik Bezděk

Poděkování

Děkuji Ing. Richardu Lipkovi, Ph.D. za cenné rady, připomínky a za čas, který mi věnoval při vedení bakalářské práce.

Abstract

This thesis focuses on the tools for automatic test generation. The thesis describes the basic types of functional and non-functional testing. The tools described in theoretical part are Conformiq Designer, Test Studio, Jubula, Agitar One, Randoop, Concordion, Jtest, EvoSuite and SoapUI. The theory also contains a description of test generation methods such as random test generation, Feedback-Directed testing, Search-based test generation, combinatorial design and test generation from static analysis, finite state machine, UML state diagram, UML activity diagram and UML communication diagram. The practical part contains detailed informations about Randoop, EvoSuite and Jtest along with samples of their usage. In the conclusion the tools are compared with each other. The best result achieved tool EvoSuite.

Key words

Software testing, automated testing, test generation, Randoop, EvoSuite, Jtest

Abstrakt

Práce se zaměřuje na nástroje pro automatické generování testů. Práce popisuje základní typy funkčních a nefunkčních testů. V teoretické části jsou popsány nástroje Conformiq Designer, Test Studio, Jubula, Agitar One, Randoop, Concordion, Jtest, EvoSuite, SoapUI a metody generování testů ze statické analýzy, z konečného stavového automatu, ze stavového UML diagramu, z UML diagramu aktivit, z UML diagramu komunikace, náhodné generování testů, Search-based generování testů, Feedback-Directed testování a kombinatorický design. Praktická část práce obsahuje bližší informace o nástrojích Randoop, EvoSuite a Jtest spolu s ukázkovými případy užití. V závěru práce jsou nástroje porovnány. Nejlepšího výsledku dosáhl nástroj EvoSuite.

Klíčová slova

Testování software, automatizované testování, generování testů, Randoop, EvoSuite, Jtest

Obsah

1 Úvod **1**

2 Testování **2**
2.1 Funkční testy 2
2.1.1 Typy funkčních testů 2
2.2 Nefunkční testy 4
2.2.1 Typy nefunkčních testů 4

3 Testovací nástroje **6**
3.1 Přehled testovacích nástrojů 7
3.1.1 Conformiq Designer 7
3.1.2 Test Studio 8
3.1.3 Jubula 9
3.1.4 Agitar One 9
3.1.5 Randoop 10
3.1.6 Concordion 11
3.1.7 Jtest 12
3.1.8 EvoSuite 12
3.1.9 SoapUI 13

4 Automatické generování testů **14**
4.1 Generování ze statické analýzy kódu 15
4.2 Náhodné generování testů 15
4.3 Generování z konečného stavového automatu 16
4.4 Search-based test generation 16
4.5 Generování ze stavového UML diagramu 17
4.6 Generování z UML diagramu komunikace 17
4.7 Generování z UML diagramu aktivit 18
4.8 Feedback-Directed Random Testing 19
4.9 Kombinatorický design 19

5 Seznámení s vybranými nástroji **21**

5.1 Randoop 21
5.1.1 Dostupnost nástroje 22
5.1.2 Generování testů 22
5.1.3 Editace vygenerovaných testů 22
5.1.4 Spouštění vygenerovaných testů 23
5.2 EvoSuite 23
5.2.1 Dostupnost nástroje 23
5.2.2 Generování testů 23
5.2.3 Editace testů 24
5.2.4 Spouštění vygenerovaných testů 24
5.3 Jtest 24
5.3.1 Dostupnost nástroje 25
5.3.2 Generování testů 25
5.3.3 Editace testů 26
5.3.4 Spouštění vygenerovaných testů 26

6 Ukázkové případy použití nástrojů 27
6.1 Ukázkový případ použití nástroje Randoop 28
6.1.1 Instalace nástroje 28
6.1.2 Generování testů 29
6.1.3 Spuštění vygenerovaných testů 32
6.1.4 Výsledky vygenerovaných testů 33
6.2 Ukázkový případ použití nástroje EvoSuite 33
6.2.1 Instalace nástroje 33
6.2.2 Generování testů 34
6.2.3 Spouštění vygenerovaných testů 37
6.2.4 Výsledky vygenerovaných testů 39
6.3 Ukázkový případ použití nástroje Jtest 39
6.3.1 Instalace nástroje 39
6.3.2 Generování testů 39
6.3.3 Spouštění vygenerovaných testů 43
6.3.4 Výsledky vygenerovaných testů 44

7 Vyhodnocení nástrojů 48
7.1 Zanesené chyby 48
7.2 Použití nástrojů 49
7.2.1 Použití nástroje Randoop 49
7.2.2 Použití nástroje EvoSuite 50
7.2.3 Použití nástroje Jtest 51
7.3 Srovnání nástrojů 51

8 Závěr **55**

1 Úvod

Rozhodl jsem se pro toto téma, protože mě zajímalo testování softwaru a automatizace samotného testování. Jsem zastáncem názoru, že testování je důležité a vývoj se bez testování v dnešní době neobejde. Díky automatizaci při spouštění testů lze kód testovat častěji v průběhu vývoje, čímž se i sníží náklady na celkový vývoj software. Zanedbání testování nebo jeho úplně vyřazení z vývoje může mít negativní dopad na výslednou kvalitu software a nespokojenost koncového zákazníka.

Cílem této práce je prostudovat nástroje na automatické testování softwaru. Začátek práce se proto zabývá metodami testování a jejich popisem. Na trhu s nástroji vyberu několik nástrojů, prostuduji jejich funkčnost a vyzkouším je. Poté bude přehled nástrojů s jejich popisem a funkcemi a nakonec udělám srovnání vybraných nástrojů.

Předně budu hledat nástroje s co největší mírou automatizace tj. automatické generování testů, generování testů podle scriptů apod. Následně sestavím tabulky hodnocení pro vybrané a otestované nástroje, ve kterých budou různé posuzovací parametry např.: úspěšnost při hledání chyb, množství nalezených chyb a pokrytí kódu. Práce neslouží k úplnému vysvětlení problematiky testování, ale jako přehled nástrojů na automatické testování.

2 Testování

V této kapitole jsou popsány typy testů, co testují, jak fungují a kdy se používají. Testy jsou rozděleny na dvě hlavní skupiny a to funkční testy a nefunkční testy. Dále jsou popsány typy jednotlivých testů, které jsou do těchto skupin zařazeny.

2.1 Funkční testy

Funkční testy se zaměřují na tzv. funkční požadavky, tedy požadavky zaměřené na funkčnost aplikace. Zákazník nebo zadavatel vývoje aplikace v nich v podstatě definuje způsob, jakým bude aplikace používána. Proto jsou také tyto požadavky obvykle zpracovány do tzv. způsobů užití (UseCase). Z těchto způsobů užití následně vycházejí testy, které se snaží prověřit správné chování aplikace ve všech situacích, které s definovanými způsoby užití mohou souviset.[17]

Funkční testy mohou být prováděny buď manuálně nebo s pomocí nástrojů. U manuálních funkčních testů jsou všechny situace a všechny stavy aplikace vytvářeny přímo samotným testerem a to často stejným způsobem, jakým bude danou aplikaci používat koncový uživatel. Vytváření testů manuálně je ovšem časově náročné, údržba testů není jednoduchá a tester může do kódu testů zanést další chyby. Za určitých podmínek je možné využívat pro vytváření, spouštění a/nebo vyhodnocování funkčních testů různé nástroje. V tu chvíli mluvíme o větší či menší míře automatizace testování.[17]

2.1.1 Typy funkčních testů

Unit testy

Jednotkový test obvykle testuje pouze danou konkrétní jednotku (kus kódu). V ideálním případě by měl být každý testovaný případ nezávislý na ostat-

ních. Při testování se snažíme testovanou část izolovat od ostatních částí programu. Za tím účelem se někdy vytvářejí pomocné objekty, které simulují předpokládaný kontext, ve kterém testovaná část pracuje (mock object).[3]

Jde o první fázi testování, která se zaměřuje na nejmenší testovatelné části aplikace. Jde obvykle o testy jednotlivých komponent aplikace na úrovni modulů, objektů a tříd. [4]

Integrační testy

U integračních testů je nutné rozlišovat mezi integrací vnitřní, která spočívá ve vzájemné komunikaci jednotlivých částí aplikace (modulů) a vnější, kdy jde o propojování jednotlivých aplikací do větších funkčních celků. U obou těchto integrací je nutné provádět integrační testy.[4]
Integrační testy se tedy zaměřují na korektní komunikaci jednotlivých modulů, resp. aplikací. Právě integrace je z pohledu vývoje aplikací poměrně kritickou oblastí a proto také mají integrační testy svoje nezastupitelné místo.[4]

Integrační testy obvykle postupují od jednotlivých modulů směrem k větším celkům. Nejdříve jsou tedy testována rozhraní jednotlivých modulů. Mnohdy jsou v této fázi využívány tzv. fake moduly. Jde o simulátory, jejichž úkolem je napodobovat komunikaci ostatních modulů ale bez jejich funkčnosti. Díky nim se ověří, že modul umí korektně odesílat i přijímat komunikaci s dalšími moduly.[4]

Další fází je spojování modulů do větších celků, které mají z pohledu testování smysl. Poslední fází je testování kompletní aplikace.[4]

Z pohledu integrace, kterou jsme si tu označili jako vnější, je nutné vzít v úvahu, že často dochází k integraci aplikací od různých výrobců. Tento druh testování je tak často náročný na kooperaci. Musí totiž při ní spolupracovat různé vývojové týmy. Ať už je tato spolupráce pouze na úrovni předání informací (popis interface) nebo jde o přímou kooperaci při testování.[4]

Smoke testy

Předtím, než je spuštěno testování aplikace, je dobré ověřit, že tato aplikace je vůbec k testování vhodná. Tím je myšleno především to, že aplikace je nainstalovaná, spuštěná, přístupná a nakonfigurovaná pro potřeby testů. Jde o rychlé testy, obvykle obsahující jednoduchý "průchod" skrz aplikaci, které dokáží ověřit všechny zmíněné atributy.[4]

Systémové testy

Jde o ověření, že aplikace jako celek funguje správně. Testuje se, že správně plní úlohu, pro kterou byla vyvinuta, že vrací správné výstupy, že byly ošetřeny všechny nestandardní situace a v neposlední řadě, že byly pokryty všechny požadavky ze strany zákazníka. Systémové testy obvykle probíhají v několika kolech. Jsou hlášeny nalezené chyby, ty jsou opraveny a v následujících kolech retestovány.[4]

2.2 Nefunkční testy

Tyto testy lze popsat jako testy všech vlastností aplikace, které přímo nesouvisí s jejími funkcemi, ale zároveň jsou podstatné pro její správné fungování. Řadí se sem především výkonové testování (Performance testing), které má ověřit například to, že aplikace i pod zátěží (větší počet současně pracujících uživatelů, větší objem dat, atd.) bude pracovat dostatečně rychle.[2]

2.2.1 Typy nefunkčních testů

Akceptační testy

Jedná se o testy na straně zákazníka. Ty jsou často prováděny podle připravených scénářů, které společně připravil zákazník s dodavatelem. Testy probíhají na testovacím prostředí u zákazníka. V této úrovni je zřejmě nejdůležitější, definovat si předem jakou formou bude probíhat oznamování chyb od zákazníka a jak zabezpečit opravení těchto chyb v co možná nejkratší době.[5]

Zátěžové testy

Jedná se o testy zátěže aplikace, případně stroje, na kterém aplikace běží. Testuje se, kolik uživatelů může přistupovat najednou, aniž by to narušilo běh aplikace [6] a ověření, že hardware stroje, kde aplikace běží, zátěž vydrží.

Výsledky těchto testů závisí na výkonu stroje, na kterém je aplikace nasazena. Zátěžové testy by tedy měly běžet na stroji, který má srovnatelný

hardware se strojem, na kterém aplikace poběží. Obvykle se testy spouští na nenasazené aplikaci,kde žádní reální uživatelé nejsou, a testy uživatele simulují.

Penetrační testy

Penetrační test je v informatice metoda hodnocení zabezpečení počítačových zařízení, systémů nebo aplikací. Provádí se testováním, simulací možných útoků na tento systém jak zevnitř, tak zvenčí. Cílem penetračního testu není vyřešit bezpečnostní problémy, ale jistým způsobem prověřit, zhodnotit úroveň zabezpečení a podat souhrnnou zprávu, a to jak na úrovni technických (nastavení otevřených portů, verze systému), tak i organizačních opatření (podvod a sociální inženýrství skrze zaměstnance).[7]

Regresní testy

Tyto testy mají za úkol ověřit, že zásahy do aplikace nebyla narušena správná funkce těch částí, které těmito zásahy neměly být ovlivněny. Jinak řečeno testuje se, že oprava chyby nebo přidání nové funkčnosti nezpůsobily novou chybu v již funkčních částech aplikace.

Regresní testy se často automatizují ve smyslu automatického spouštění testů, protože u nich jde o opakované provádění stejných operací se známým výsledkem. Jejich cílem je tak zjistit, zda neexistují odchylky mezi výstupem získaným před zásahem do aplikace a výstupem po tomto zásahu.[4]

Uživatelské testy

Princip spočívá v tom, že tato metoda pozoruje chování samotných uživatelů, čímž může odhalit chyby, které zůstaly vývojářům skryté. Toto testování je vhodné aplikovat již při vývoji aplikace, čímž může vývojářům poskytnout cenné informace, se kterými snáze eliminují další problémy. Uživatelské testování tak může určovat správný směr při vývoji samotné aplikace a předcházet tak vznikajícím problémům.[7]

3 Testovací nástroje

Nástrojů na testování je na trhu mnoho a nelze tedy porovnat úplně všechny. Podíváme se na přehled nástrojů, z nichž některé poté podrobíme bližší analýze a praktickému testování. Při hledání nástrojů jsem bral ohled na několik charakteristik.

- Licence
- Dostupnost
- Cena
- Stáří nástroje/verze
- Typy testů
- Podporovaná technologie
- Druh automatizace

Pojem druh automatizace se ovšem u nástrojů liší. U některých je považován tento termín za automatické spouštění testů, které napsal tester, někde za generování testů samotným nástrojem nebo generování testovacích skriptů. Tato práce se zaměřuje spíše na nástroje generující testovací kódy nebo generující testovací skripty.

3.1 Přehled testovacích nástrojů

3.1.1 Conformiq Designer

Charakteristika	**Popis**
Licence	Komerční
Dostupnost	Demo verze
Cena	Cena na míru podle potřeb zákazníka
Stáří nástroje	verze 4.4.3 (2012)
Typy testů	Black box, white box, unit, integrační
Podporovaná technologie	C/C++, TCL, TTCN-3, Perl, Python, Java
Druh automatizace	Generování testovacích skriptů

Tabulka 3.1: Údaje o nástroji Conformiq Designer

Conformiq Designer je nástroj pro automatické generování skriptů na základě modelu testovaného systému. To znamená, že nástroj vygeneruje automaticky testy pro systém, pokud mu je jako vstup vložen model testovaného systému. Tento model je popis očekávaného chování systému a vytváření se jako stavový model systému pomocí jazyka UML.[9]

3.1.2 Test Studio

Charakteristika	**Popis**
Licence	Komerční
Dostupnost	Trial verze
Cena	2499$, 169$/měsíc
Stáří nástroje	verze 2015.3.1314 (2016)
Typy testů	Funkční, zátěžové, výkonnostní
Podporovaná technologie	HTML, AJAX, Silverlight, ASP.NET, MVC, JavaScript, WPF
Druh automatizace	Nahrávání a spouštění testů

Tabulka 3.2: Údaje o nástroji Test Studio

Test studio slouží pro testování webových a mobilních aplikací. Nástroj umí nahrávat testy přímo v prohlížeči nebo může tester psát testovací scripty. Nahraný test převede nástroj do kódu, ve kterém je aplikace psaná a který může tester upravit nebo rozšířit podle potřeby.[10] Jednou nahraný test lze pustit v podporovaných prohlížečích bez nutnosti test znovu nahrávat. Nástroj umožňuje testy spouštět přes virtuální uživatele a lze tak vytvářet zátěžové a výkonnostní testy.

3.1.3 Jubula

Charakteristika	Popis
Licence	EULA
Dostupnost	Po registraci
Cena	Zdarma
Stáří nástroje	verze 8.2 (2016)
Typy testů	Black-box, integrační, systémové, akceptační, regresní
Podporovaná technologie	HTML, Java
Druh automatizace	Nahrávání a spouštění testů

Tabulka 3.3: Údaje o nástroji Jubula

Jubula je nástroj, který poskytuje funkční testování GUI. Testy se vytvářejí pomocí drag&drop, kdy pouze vyberete akci, která se má provést (např.: kliknutí na tlačítko, zadání vstupní hodnoty) a ta se zařadí do seznamu všech akcí, které je poté možné spustit. Není potřeba psát žádný kód a to značně urychluje práci.

3.1.4 Agitar One

Charakteristika	Popis
Licence	Komerční
Dostupnost	Trial verze
Cena	Cena na míru podle potřeb zákazníka
Stáří nástroje	verze 5.5 (2012)
Typy testů	Unit, regresní
Podporovaná technologie	Java
Druh automatizace	Generování unit testů

Tabulka 3.4: Údaje o nástroji Agitar One

Nástroj Agitar One generuje unit testy, které dokumentují chování testovaného kódu.[11] Nástroj dokáže generovat dynamické testovací případy, vytvářet vstupní data a poté analyzovat výsledky testů. Agitar One podporuje pouze programovací jazyk Java a není tedy tolik universální oproti jiným nástrojům.

3.1.5 Randoop

Charakteristika	**Popis**
Licence	MIT licence
Dostupnost	Dostupný
Cena	Zdarma
Stáří nástroje	verze 2.1.4 (2016)
Typy testů	Unit, regresní
Podporovaná technologie	Java
Druh automatizace	Generování unit testů

Tabulka 3.5: Údaje o nástroji Randoop

Randoop je generátor unit testů pro jazyk Java. Automaticky vytváří unit testy pro třídy ve formátu JUnit. Používá metodu náhodného generování testů, kdy nástroj spustí sekvenci testů, kterou vytvořil a s použitím výsledků testů zachycuje chování programu. Nástroj může být použit na hledání bugů v kódu nebo k vytvoření regresních testů, které můžou testera varovat, pokud se v kódu něco změní v budoucím vývoji.[12]

3.1.6 Concordion

Charakteristika	**Popis**
Licence	Apache Licese v2.0
Dostupnost	Dostupný
Cena	Zdarma
Stáří nástroje	verze 1.5.1 (2015)
Typy testů	Akceptační
Podporovaná technologie	Java, .NET, Python, Ruby
Druh automatizace	Vytváření akceptačních testů

Tabulka 3.6: Údaje o nástroji Concordion

Nástroj Concordion slouží k vyváření akceptačních testů a funguje na základě HTML specifikace. Tato specifikace obsahuje příklady chování programu. Concordion umožňuje psát specifikace v přirozeném jazyce s použitím odstavců, tabulek apod. Specifikace se vytváří na konkrétní případy užití s použitím příkazů `set, execute, assertEquals`, to umožňuje, aby byly příklady zkontrolovány proti skutečnému systému. Díky tomu se specifikace snadno píše, čte a pomáhá každému porozumět, co má daná funkce programu dělat. Specifikace jsou aktivní a jsou provázány se systémem, který se testuje, proto jsou vždy aktuální i pokud se v systému udělá změna.[13]

3.1.7 Jtest

Charakteristika	**Popis**
Licence	Komerční
Dostupnost	Demo verze
Cena	Cena na míru podle potřeb zákazníka
Stáří nástroje	verze 9.5.13 (2014)
Typy testů	Integrační, regresní, unit
Podporovaná technologie	C, C++, Java, .NET
Druh automatizace	Generování unit testů

Tabulka 3.7: Údaje o nástroji Jtest

Jtest automaticky analyzuje kód a poté generuje testy s vysokým pokrytím ve formě unit testů. Tyto znovupoužitelné testy zachycují existující chování programu pro regresní testy. K tomu mohou být vytvořeny i funkční testy díky "sledování" programu. Nasimuluje se chování programu, které chcete otestovat, a poté se automaticky vygenerují unit testy. Testovací scénáře mohou být spuštěny nezávisle na systému a oddělit tak změny v chování programu.[14]

3.1.8 EvoSuite

Charakteristika	**Popis**
Licence	GNU Library or Lesser General Public License
Dostupnost	Dostupný
Cena	Zdarma
Stáří nástroje	verze 1.0.3 (2016)
Typy testů	Unit
Podporovaná technologie	Java
Druh automatizace	Generování unit testů

Tabulka 3.8: Údaje o nástroji EvoSuite

EvoSuite je nástroj, který automaticky generuje testovací případy pro třídy napsané v jazyce Java. EvoSuite používá hybridní přístup, který generuje a optimalizuje celé testovací případy s ohledem na uspokojení kritérií pokrytí kódu. Nástroj zachycuje chování programu a umožňuje programátorovi detekovat odchylky od očekávaného chování s cílem ochránit program od budoucích defektů, které by mohli očekávané chování narušit.[15]

3.1.9 SoapUI

Charakteristika	**Popis**
Licence	EUPL
Dostupnost	Dostupný
Cena	Zdarma
Stáří nástroje	verze 5.2.1 (2014)
Typy testů	Funkční, regresní
Podporovaná technologie	SOA, REST
Druh automatizace	Vytváření a spouštění testů

Tabulka 3.9: Údaje o nástroji SoapUI

Jedná se o nástroj testující webové služby pro SOA a REST. Nástroj podporuje automatické testování podle scénářů. Lze vytvořit testovací případy jako dlouhou sekvenci volání v aplikaci. Nástroj lze použít například na přihlášení se do aplikace, získání požadované hodnoty, která se poté vrátí jako parametr pro další volání v aplikaci, apod. Při testování se dají použít i reálná data k řízení testů a lze tak zjistit jestli vše funguje jak má.

4 Automatické generování testů

Testování je běžná technika k zajištění požadovaného chování softwaru. Testovací případy mohou být napsány před samotnou implementací programu a slouží pak jako specifikace (tzv. test-driven development [27]). Mohou být napsány tak, aby přímo testovali daný program s cílem najít v něm chyby nebo mohou být psány tak, aby zachycovali chování programu a ochránili tak program před budoucími chybami při dalším vývoji. Nicméně psaní samotných testů může být obtížné a ručně psané testovací případy nebo testovací plány nebývají z pravidla kompletní. Proto výzkumníci zkoumají možnost automatického generování testů tak, aby mohli nahradit manuálně psané testy.[25]

Automatické generování testů u tzv. black-box testování se skládá ze dvou částí: 1) vygenerovat vhodná vstupní testovací data a 2) určení zda vygenerované testy našli nějaké chyby, to znamená, že vygenerované testy potřebují nějakého pozorovatele, který určí, zda je test vhodný nebo ne.[25] Pozorovatelem může být buď člověk, který ručně vybere vhodné testy, nebo nástroj, který provede stejnou činnost na základě algoritmu.

Automatické generování testovacích případů je zvláště vhodné, když existují i automatičtí pozorovatelé a hledání chyb se stane kompletně automatickou záležitostí, která nevyžaduje zásah člověka.[25]

Při testování softwaru se často používá termín pokrytí kódu. Pokrytí kódu je měřítko, které procentuálně vyjadřuje kolik zdrojového kódu bylo spuštěno při testování. Existuje několik kritérií pokrytí jako například pokrytí příkazů/řádků(kontroluje se zda byl příkaz/řádek spuštěn při průběhu testu), pokrytí větví(kontroluje se zda se otestovalo kladné i záporné vyhodnocení podmínky), aj.[36]

V této kapitole popíši stručně několik metod na generování testů. V praxi se pak metody používají samotné, tak jak fungují, nebo jsou součástí jiných metod, které je dále rozvíjejí.

4.1 Generování ze statické analýzy kódu

Statická analýza kódu je analýza počítačového softwaru, která se provádí bez spuštění testovaného programu. U většiny případů je analýza prováděna na zdrojovém kódu programu a v některých případech na nějaké formě objektového kódu programu. Od statické analýzy se očekává, že odhalí mnoho běžných problémů v kódu před tím, než je program vydaný.[24]

Mezi typy statické analýzy můžeme zařadit například manuální kontrolu kódu, kterou provádí člověk. Je to ovšem velmi časové náročné a aby byla kontrola efektivní, měl by kontrolor vědět, jaké typy chyb má hledat. Statická analýza kódu prováděná člověkem se dá ještě dále rozdělit do dvou kategorií: vlastní kontrola, kterou provádí programátor při vývoji softwaru a kontrola třetí stranou.[24]

Při vývoji software se ovšem upřednostňuje použití nástrojů na statickou analýzu, protože jsou rychlejší a lze tedy kód testovat častěji. Statickou analýzu provádí i překladač programového kódu a ověřuje jestli program splňuje požadovanou syntaxi.

4.2 Náhodné generování testů

Náhodné testování funguje tak, že se generují náhodně vstupní data ze vstupního stavového prostoru. Je tedy jasné, že tato metoda má značné nevýhody. Při příliš velkém stavovém prostoru nelze otestovat všechny vstupy a vygeneruje se příliš mnoho testů, z nichž může být velký počet testů redundantních. Problém může nastat i při analýze výsledků, pokud nebude použit program k vyhodnocení a o správnosti a použitelnosti testů bude muset rozhodnout člověk. Nad vyhodnocováním výsledků se poté stráví mnohonásobně více času než při generování testů.

Výhoda této metody je její použití při malém stavovém prostoru. Dá se poté očekávat, že se testy "trefí" na většinu vstupů, otestují program dostatečně a generování testů bude méně časově náročné než při manuálním psaní.

I když se zdá, že technika není příliš využitelná v praxi, používá často právě pro její element náhodnosti, například u zátěžových testů serverů. Dále se používá také jako základ pro rozvíjení dalších metod, například Adaptive Random Testing([29]), Directed Random Testing([30]), aj.

4.3 Generování z konečného stavového automatu

Konečný automat zachycuje chování software nebo software-hardware systému jako konečnou sekvenci stavů a přechodů mezi nimi. Chování modelovaného systému pro danou sekvenci vstupů může být dosaženo tím, že spustíme automat v jeho počátečním stavu a provedeme sekvenci.[16]

Pro generování testů z konečného stavového automatu existuje několik metod, jako například W-metoda, částečná metoda označovaná jako Wp-metoda, unique input/output metoda aj.[16]

Detailní popis těchto metod je nad rámec této práce, ale postupy a informace o těchto metodách spolu s příklady jsou uvedeny zde: [16].

4.4 Search-based test generation

Před popisem techniky si zavedeme několik použitých pojmů pro lepší porozumění popisu techniky.

- Populace - testovací případy
- Vhodnost - jak blízko je kandidát k řešení a uspokojení určitého typu (kritéria) pokrytí kódu
- Mutace - náhodná změna části testovacího případu
- Křížení - prohození částí testovacích případů mezi sebou

Search-based testing popisuje použití vhodného vyhledávacího algoritmu, který má za úkol generování testovacích případů. Jeden z běžně využívaných algoritmů je tzv. genetický algoritmus.

Genetický algoritmus se snaží imitovat přírodní procesy evoluce. Počáteční populace většinou náhodně vybraných kandidátů na řešení je vyvinuta s použitím vyhledávacích operátorů, kteří ale připomínají více eugeniku než evoluci. Eugenika je sociálně-filosofický směr zaměřený na studium metod, které povedou k dosažení co nejlepšího genetického fondu člověka.[28] V případě generování testů to znamená, že vybrání rodičů pro reprodukci se provádí na základě největší "vhodnosti". Funguje je to zjednodušeně tak, že se

kandidáti ohodnotí číslem a ti s malou hodnotou se odstraní. Reprodukce se dále provede křížením a mutací s určitou pravděpodobností. S každou iterací genetického algoritmu se vhodnost populace zvyšuje nebo snižuje, podle nastavení parametrů mutace a křížení. Iterace se provádí tak dlouho, než se nenajde suboptimální řešení. Genetický algoritmus bohužel optimální řešení nemusí poznat a tak tento úkol přebírá metoda, která genetický algoritmus využívá.[26]

Tradiční přístup search-based testování je optimalizovat testovací případ pro každé kritérium pokrytí v izolaci. Generování celých testovacích plánů pomáhá optimalizovat všechny testovací případy tak, aby splňovaly kritéria pokrytí, místo toho, aby se uvažovalo nad vytvořením různých testovacích případů pro různá kritéria pokrytí. Znamená to, že výsledek není negativně ovlivněn ani podle pořadí, ani podle obtížnosti nebo neproveditelnosti jednotlivých kritérií pokrytí.[26]

4.5 Generování ze stavového UML diagramu

Pro použití této techniky je nejdříve zapotřebí sestavit UML stavový diagram. Tyto diagramy poskytují široký náhled, který popisuje kompletní funkce systému nebo aplikace, protože diagramy zachycují každý stav systému. Použití diagramu napomáhá k snížení možnosti výskytu chyb a neočekávaného chování systému, protože jsme nuceni vzít v úvahu všechny možnosti, na které musí systém reagovat.[19]

Poté se stavový diagram převede do stavového grafu. V grafu jsou stavy reprezentovány jako uzly a přechody mezi stavy jako hrany. Lze předpokládat, že v grafu existuje unikátní počáteční stav a jeden nebo více koncových stavů.[19] Z tohoto stavového grafu se vytvoří konečný stavový automat a z automatu se poté vytvářejí samotné testy.

Není zde uveden detailní popis techniky. Detailní informace o této technice spolu s příkladem generování jsou uvedeny zde [19].

4.6 Generování z UML diagramu komunikace

V diagramech UML je interakce jednotkou chování, která se zaměřuje na výměny informací mezi objekty, které lze pozorovat. Interakce tedy reprezentuje komunikaci mezi objekty.[20]

Diagram komunikací ukazuje, jak spolu objekty vzájemně komunikují, aby dosáhli požadovaného chování. Oproti sekvenčnímu diagramu ukazuje diagram komunikace strukturální vztahy mezi objekty nebo životními linkami (*"lifelines"*). Životní linka reprezentuje individuálního účastníka v interakci. Na druhou stranu diagram komunikace neukazuje čas jako oddělenou dimenzi. Sekvence zpráv a organizace ve vláknech musí být rozhodnuty s použitím sekvenčních čísel.[20]

Po vytvoření diagramu komunikace se sestaví komunikační strom. Komunikační strom reprezentuje možnou sekvenci zpráv v interakci. Zpráva, která začíná komunikaci, se označí jako kořen stromu a listy stromu odpovídají konci sekvence zpráv. Z toho stromu se poté iterativně zvolí predikáty.[20] Predikát je jazykové sdělení nebo výraz, o němž má smysl tvrdit, že je pravdivý nebo nepravdivý.[21] Poté se predikát přetvoří tak, aby se našla testovací data odpovídající predikátu. Pro každý zvolený predikát se následně zaznamená testovací případ a je zvolen nový predikát. Zvolení predikátu a vygenerování testovacích případů se opakuje tak dlouho, dokud nejsou vybrány všechny predikáty pro generování testů.[20]

Kompletní popis metody s detailními popisy přetváření predikátů a generování testovacích případů a následnou implementací jsou uvedeny zde: [20].

4.7 Generování z UML diagramu aktivit

Diagram aktivit se používá pro popis dynamických aspektů systému. Znázorňuje tok řízení z aktivity do aktivity. Používá se také k modelování obchodních procesů a workflow. Diagram aktivit se soustřeďuje spíše na proces výpočtu než na objekty účastnící se výpočtu (i když i objekty mohou být znázorněny jako prvek aktivity). Diagramy stavů a diagramy aktivit jsou si podobné (oba ukazují sekvenci stavů, které nastávají v čase, a ukazují podmínky způsobující přechody mezi stavy). Rozdíl mezi těmito diagramy je však v tom, že diagram stavů se soustřeďuje na stavy objektu (tj. objektu provádějícího výpočet či objektu, se kterým je výpočet prováděn), kdežto diagram aktivit se zaměřuje na stav samotného výpočtu (stav procesu, algoritmu, ...), kde může být účastno i více objektů a kde jsou znázorněny řídící a informační toky mezi prvky diagramu.[23]

Prvním krokem pro použití této metody je sestavení diagramu aktivit, ze kterého se následně vytvoří graf aktivit. Graf aktivit je orientovaný graf, kde každý uzel reprezentuje elementy systému (počáteční uzel, konečný uzel,

rozdělovací uzel, spojovací uzel, podmínky, ...) a každá hrana reprezentuje datový tok mezi uzly. Poté se použije algoritmus na procházení grafem a vytvoří se cesty aktivit (posloupnost uzlů podle případu užití systému). Z těchto cest jsou dále generovány testovací případy.[22]

Kompletní popis techniky s detaily vytváření grafů aktivit a cest aktivit spolu s názorným příkladem je uveden zde: [22].

4.8 Feedback-Directed Random Testing

Tato technika je vylepšením techniky náhodného generování dat tím, že zahrnuje zpětnou vazbu získanou ze spouštění testovacích vstupů, které jsou průběžně vytvářeny. Technika vytváří vstupy přírůstkově náhodným voláním metod a hledáním argumentů z dříve vytvořených vstupů. Hned, jak je vstup vytvořen, zkontroluje se. Výsledek kontroly určí, zda je vstup redundantní, nedovolený, neplatný nebo použitelný pro další generování vstupů. Výstup této techniky je testovací plán, který obsahuje unit testy pro testované třídy. Technika se používá na generovaní unit testů pro objektově orientované programy.[18]

4.9 Kombinatorický design

Softwarové aplikace jsou často vytvářeny tak, aby pracovaly v různých prostředích. Kombinace faktorů, jako jsou operační systém, síťové připojení a hardware, vede k mnoha různých kombinacím prostředí. K zajištění vysoké spolehlivosti napříč prostředími je potřeba aplikaci otestovat na co největším počtu různých prostředích. Počet těchto prostředí však může být obrovský a je tedy nemožné aplikaci otestovat na všech.[16]

Podobná situace platí i pro generování testů. Nelze vždy otestovat úplně všechny možnosti, pokud je systém rozsáhlý. Stačí ale otestovat dostatečně malou množinu kombinací, která pokryje co největší počet těchto možností.

Techniky pro generování zmenšených množin testů jsou uvedeny zde: [16].

	Nástroje								
Metody	Conf	TStudio	Jubula	AgitarO	Randoop	Conc	JTest	EvoS	SoapUI
Statická analýza				X			X		
Náhodné generování									
Konečný stavový automat									
Search-based								X	
Stavový UML	X								
UML aktivit									
UML komunikace									
Feedback-directed					X				
HTML specifikace						X			
Nahrávání testů přes GUI		X							
Drag&Drop generování přes GUI			X						
Podpora manuálního psaní testů									X

Tabulka 4.1: Srovnání metod generování testů

5 Seznámení s vybranými nástroji

V této se blíže seznámíme s třemi vybranými nástroji, které budeme zkoušet a porovnávat. Tato práce je zaměřena na automatické generování testů a proto jsem vybral nástroje Randoop, EvoSuite a Jtest. Všechny nástroje slouží k automatickému generování unit testů, ale každý na to používá jinou metodu generování(viz. tabulka 4.1). U nástrojů budu popisovat několik mnou vybraných kritérií(viz. tabulka 5.1) a popíši i práci s každým nástrojem.

Kritérium	**Popis kritéria**
Dostupnost nástroje	Podmínky pro získání nástroje
Generování testů	Jakým způsobem se v nástroji generují testy
Editace testů	Jakým způsobem lze testy editovat
Spouštění testů	Jakým způsobem lze testy spouštět

Tabulka 5.1: Kritéria popisu nástrojů

5.1 Randoop

Nástroj Randoop generuje jednotkové testy s použítm metody feedback-directed random test generation. Nástroj může být použit dvěma způsoby: hledání chyb v programu a vytvoření regresních testů na kontrolu softwaru při budoucím vývoji. Randoop se použil na odhalení dříve neobjevených chyb v knihovnách programů od společností Sun, IBM a v jádru komponent .NET. Randoop využívá například společnost ABB a jeho vývoj pokračuje i nadále.[12] Kromě generování testů pro jazyk Java lze program použít i pro programovací jazyk C#.

5.1.1 Dostupnost nástroje

Nástroj Randoop je vyvíjen jako open-source program. Lze ho stáhnout zdarma jako spustitelný balíček na stránkách projektu, a spouštět přes příkazovou řádku, nebo jako plug-in do vývojových prostředí Eclipse a NetBeans.

5.1.2 Generování testů

Pro vytváření testů pomocí nástroje Randoop je potřeba mít zdrojové kódy programu, pro který chceme testy generovat. Při použití spustitelného balíčku slouží k ovládání nástroje příkazová řádka. Po zadání příslušných parametrů nástroj vytvoří několik souborů, z nichž jeden slouží pro nastavení generování testů a další už jsou samotné vygenerované testy. Nástroj lze použít i v případě, že chceme generovat testy pro více souborů se zdrojovými kódy. Stačí vytvořit textový soubor s názvy tříd (název každé třídy na nové řádce) a název souboru vložit jako parametr v příkazové řádce spolu s odpovídajícím příkazem. Tento způsobo lze použít i pokud potřebujeme generovat testy jen pro určité třídy a metody ve zdrojových kódech.

Pokud se rozhodneme použít plug-in například ve vývojovém prostředí Eclipse, je práce s nástrojem ještě jednodušší. Generování testů probíhá tak, že vybereme třídy, balíčky nebo celý projekt a spustíme je jako vstupní data pro plug-in Randoop. Při zvolení celého projektu se nástroj zeptá, pro jaké balíčky, popřípadě třídy chceme testy vygenerovat a také kolik testů má nástroj vygenerovat nebo po kolika vteřinách generování testů má nástroj přestat. Vygenerované testy se uloží do nové složky v projektu a jsou spustitelné pomocí knihovny JUnit. Nástroj používá pro výstupní soubory svoje vlastní názvy, které se ovšem neliší v závislosti na testovaných souborech. Názvy lze změnit při konfiguraci generování testů, ale je možné si přepsat jednou vygenerované testy novými, které jsou generované pro jinou třídu, pokud zapomeneme názvy přepsat.

5.1.3 Editace vygenerovaných testů

Nástroj generuje testy jako externí soubory v programovacím jazyce Java a lze pro jejich editaci použít jakýkoliv textový editor, nejlépe však nějaký co umí rozeznat syntaxi jazyka Java pro lepší přehlednost. Při použití plug-inu ve vývojovém prostředí lze testy editovat přímo v něm.

5.1.4 Spouštění vygenerovaných testů

Nástroj Randoop jako takový neumí testy spouštět, ale pouze generovat. Ke spuštění testů slouží nejlépe nástroj JUnit. Vygenerované testy lze spouštět neustále a kontrolovat tak, jestli se nezměnilo chování programu.

5.2 EvoSuite

EvoSuite automaticky generuje unit testy pro třídy napsané v programovacím jazyce Java a pracuje na úrovni byte kódu Javy. Ke generování testů není zapotřebí zdrojových kódů. EvoSuite je založen na search-based testování a používá ke generování testů genetický algoritmus. EvoSuite se zaměřuje na generování celých testovacích scénářů (testovací scénář je kolekce testovacích případů) zaměřujících se na více kritérií pokrytí najednou. Vývoj nástroje je stále aktivní s poslední vydanou verzí na konci února roku 2016.

5.2.1 Dostupnost nástroje

Nástroj EvoSuite je vyvíjen jako open-source nástroj pod licencí GNU Lesser General Public License([32]) a je zdarma ke stažení na stránkách projektu jako samostatný spustitelný balíček, který lze spouštět přes příkazovou řádku. Další možností je stáhnout nástroj jako plug-in do vývojových prostředí Eclipse a IntelliJ IDEA. Na stránkách projektu je uvedeno, že plug-in do IntelliJ IDEA je stabilní, ale plug-in do Eclipsu je zatím experimentální a není zaručeno, že bude fungovat tak, jak má.

5.2.2 Generování testů

Nástroj EvoSuite generuje testy z byte kódu a nepotřebuje k práci zdrojové soubory. Pokud použijeme spustitelný balíček, lze nástroj pustit z příkazové řádky s příslušnými parametry. EvoSuite byl ze začátku vyvíjen pro operační systém s jádrem Linux a je možné, že bude mít problémy se souborovými cestami v jiných operačních systémech. Při generování testů vytvoří nástroj dva soubory, jeden pro nastavení parametrů testování a druhý, který obsahuje už samotné unit testy. Při spuštění nástroje z příkazové řádky lze ge-

nerovat testovací případy pro více tříd najednou. Plug-in v Eclipsu bohužel generování pro více tříd najednou neumožňuje a testy jde generovat jen pro jednotlivé třídu. Vygenerované testy uloží do nově vytvořené složky a testy jsou strukturovány v balíčcích stejně tak, jako byly v projektu. Názvy vygenerovaných testů se shodují s třídami, z nichž vznikly, a to přispívá celkové přehlednosti.

5.2.3 Editace testů

EvoSuite generuje testy jako soubory v programovacím jazyce Java ve formátu JUnit. Pro jejich editaci lze použít textový editor (pro snazší práci nějaký, co umí rozeznat syntaxi jazyka Java) nebo můžeme testy editovat přímo v Eclipsu při použití plug-inu na jejich vygenerování. Vygenerované testy jsou tvořeny velmi čitelně, není tedy problém se v nich na první pohled vyznat a upravit je.

5.2.4 Spouštění vygenerovaných testů

Při použití plug-inu v Eclipsu závisí vygenerované JUnit testy na EvoSuite frameworku. Tento framework musí být zařazen k projektu, když jsou testy kompilovány a spouštěny. Je potřeba přidat balíček `evosuite-standalone-runtime.jar` soubor do projektu. Na spouštění testů lze použít i balíček `evosuite.jar`. Jedná se o nadstavbu standalone verze, ale tento balíček přidává do projektu i knihovny, které nepotřebujeme.[31]

5.3 Jtest

Nástroj Parasoft Jtest je ucelené řešení pro automatizování velké škály praktik, které slouží ke zlepšení vývoje a kvality software a týmové produktivity. Zaměřuje se na validaci Java kódu a aplikací.
Jtest napomáhá k:

- Statické analýze: Statická analýza kódu, statická analýza data flow a analýza metrik

- Revizi kódu: Příprava, upozornění a sledování
- Unit testování: Vytváření JUnit testů, spouštění testů, optimalizace a údržba
- Detekci chyb při běhu programu: Podmínky, výjimky, spotřeba zdrojů a kontrola práce s pamětí, kontrola slabých míst v zabezpečení, a další

Nástroj poskytuje týmům praktickou cestu k předcházení, odhalování a opravování chyb, aby bylo zajištěno, že se jejich Java kód chová tak, jak má. Na podporu rychlé nápravy má každý nalezený problém prioritu založenou na konfigurovatelném přiřazování závažnosti. Automaticky se problém přiřadí k vývojáři, který napsal příslušný kód, a pošle se mu zpráva do jeho vývojového prostředí s přímým odkazem na problematický kód s popisem, jak ho opravit.[34] Jakožto komerční nástroj je Jtest více přizpůsoben k celkovému vývoji software a neslouží pouze ke generování unit testů.

5.3.1 Dostupnost nástroje

Parasoft Jtest je komerční nástroj a nelze ho volně stáhnout a používat. Na stránkách produktu si lze domluvit demo ukázku použití Jtestu a po domluvě i trial verzi. Jtest má dvě verze, a to plug-in do vývojového prostředí Eclipse nebo svoje vlastní vývojové prostředí, které je na Eclipse založeno.

5.3.2 Generování testů

Vytváření testů a práce s nástrojem probíhá přes tzv. testovací konfigurace. Jedná se o funkci, kde si lze navolit například to, co chceme kontrolovat statickou analýzou, jestli chceme generovat jednotkové testy a nastavení generování testů, jaké typy pokrytí chceme zachytit, možnost reportování nalezených chyb podle závažnosti, aj.

Jtest automaticky generuje testovací případy ve formátu JUnit vzhledem k parametrům definovaným v testovací konfiguraci. Jtest generuje jeden soubor s testovací třídou pro každou testovanou třídu. V každé testovací třídě je alespoň jedna testovací metoda pro každou metodu v testované třídě spolu s doplňujícím kódem pro test. Náhradní objekty jsou doplněny, pokud testovaná třída obsahuje reference na externí zdroje, jako je například databáze,

knihovny třetích stran, vstup/výstup souborového systému, síťové vstupy-/výstupy, atd. Pokud máme vygenerované testy pro danou třídu a chceme je generovat znovu, Jtest přidá nové testovací případy do existující testovací třídy.[34] Při použití základního nastavení se testy vygenerují do nového projektu a jsou strukturovány jako projekt, který obsahoval testovanou třídu. Balíčky v projektu obsahují jeden soubor s konfigurací testů a soubory se samotnými unit testy. Názvy testovacích tříd jsou shodné s testovanými třídami a díky struktuře, ve které jsou testy generovány, je vše přehledné a snadno dohledatelné.

5.3.3 Editace testů

Nástroj Jtest vytváří testy jako nové soubory v programovacím jazyce Java. Pro jejich editaci lze použít libovolný textový editor a pro lepší přehlednost kódu zvolit takový, který umí rozpoznat syntaxi jazyka Java. Při použití pluginu nebo samostatné verze jsou testy lehce editovatelné přímo ve vývojovém prostředí, ve kterém jsou vygenerovány.

5.3.4 Spouštění vygenerovaných testů

Jtest generuje testy ve formátu JUnit a je potřeba mít knihovnu JUnit připojenou k projektu. Testy lze poté spouštět jednoduše z vývojového prostředí a v přehledném formátu dostat výsledky.

6 Ukázkové případy použití nástrojů

V této kapitole ukážu praktické použití nástrojů na aplikaci, kterou jsem vytvořil. Aplikaci jsem napsal jako jednu jednoduchou třídu, ze které se vygenerují unit testy ve všech nástrojích. Následně do aplikace zanesu několik chyb. Cílem je otestovat, jestli nástroje zanesené chyby odhalí, nebo ne. Chyby, které se budou nástroje snažit odhalit jsou:

- Vrácení null z metody místo očekávaného objektu
- Vyhození výjimky před dokončením metody
- Vstupní hodnota parametru mimo očekávané meze
- Vrácení nesprávné hodnoty
- Přepsání vnitřku metody

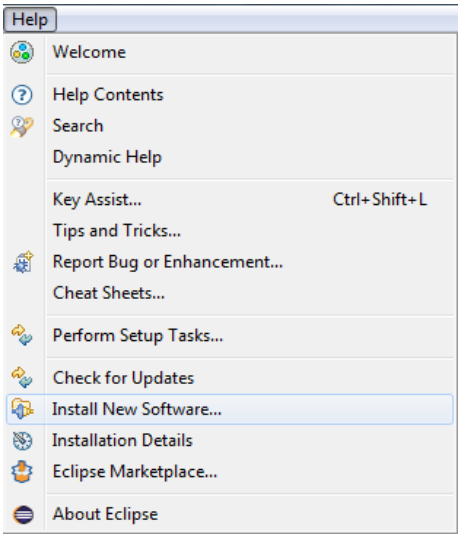
Kromě vyhodnocování výsledků jednotlivých nástrojů jsou ukázkové případy psány jako obecný návod, jak nástroj nainstalovat, spustit a obsluhovat. Případy jsou psány tak, aby je mohl použít i ten, kdo s nástroji nikdy nepracoval. Ukázky jsou pro snadnější orientaci při ovládání nástrojů doplněny obrázky. Dále zde jsou uvedeny jako příklady pro srovnání vygenerované testy u jednotlivých nástrojů. Je z nich vidět, že ačkoliv dělají nástroje to samé, ne vždy musí být výsledek stejný. Popisy nástrojů a návody jsou psány pro vývojové prostředí Eclipse, protože ho lze použít pro testování všech vybraných nástrojů.

6.1 Ukázkový případ použití nástroje Randoop

Randoop lze použít jako plug-in do Eclipse nebo jako samostatný spustitelný balíček. Ukázka použití je doplněna o obrázky z vývojového prostředí a popis základních příkazů při použití samostatného balíčku.

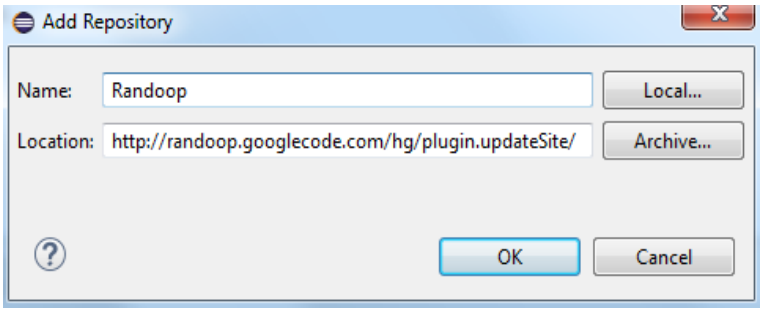
6.1.1 Instalace nástroje

Při použití spustitelného balíčku stačí na stránkách:
`https://github.com/randoop/randoop/releases/tag/v2.1.4`
stáhnout soubor `randoop-2.1.4.jar` a začít ho ihned používat. Pokud chceme požít plug-in do vývojového prostředí Eclipse (je doporučeno mít vždy nejnovější verzi) je instalace nástroje snadná. Po spuštění Eclipse stačí v navigační liště v záložce "Help" zvolit položku "Instal New Software..." (viz. obr. 6.1). Kliknutím na tlačítko "Add..." se zobrazí se nové okno, kde zadáme libovolný



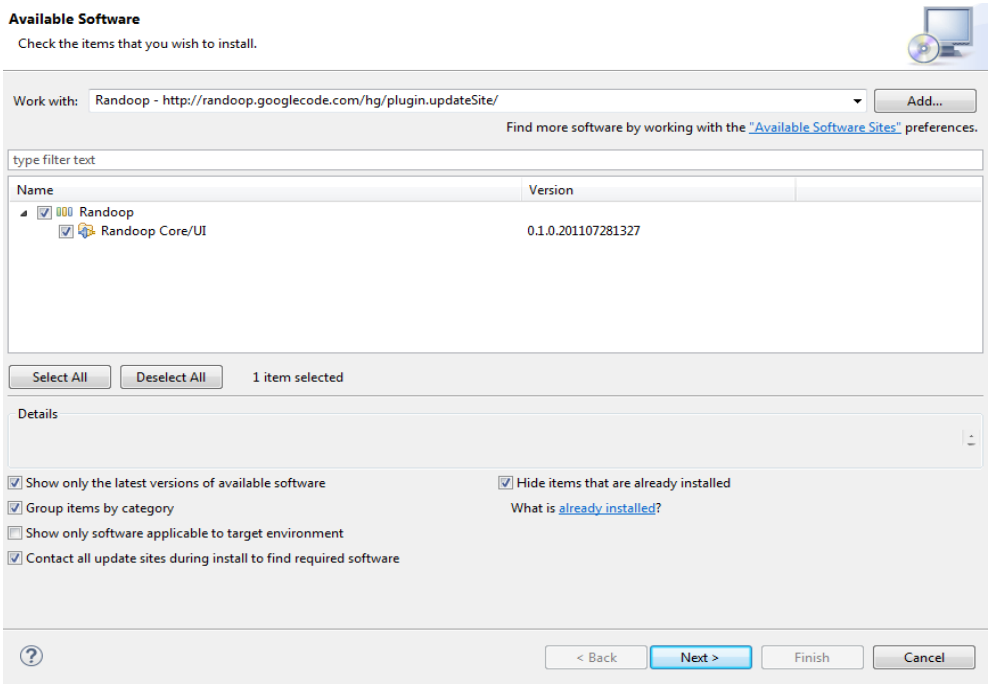
Obrázek 6.1: Instalace nástroje Randoop

název, pod kterým se bude stránka plug-inu zobrazovat a adresa
`http://randoop.googlecode.com/hg/plugin.updateSite/` (viz. obr. 6.2). Po potvrzení formuláře se zobrazí zaškrtávací seznam s nástrojem Randoop. Zaškrtnutím políčka s nástrojem a zvolením "Next >" (viz. obr. 6.3) nástroj



Obrázek 6.2: Instalace nástroje Randoop

nainstalujeme a po restartování vývojového prostředí je připraven k použití.

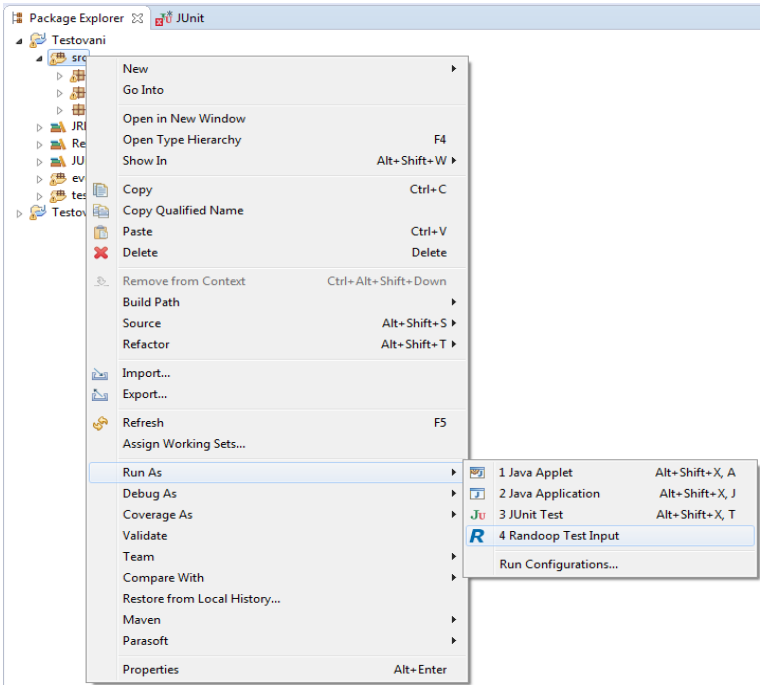


Obrázek 6.3: Instalace nástroje Randoop

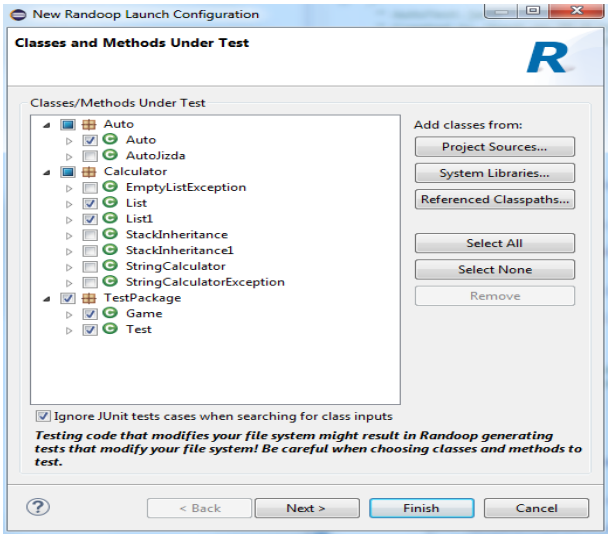
6.1.2 Generování testů

Při použití vývojové prostředí Eclipse se testy generují jednoduše tak, že v záložce "Package Explorer", kde je seznam projektů, balíčků, tříd, atd., vybe-

reme položku, kterou chceme otestovat, klikneme pravým tlačítkem myši a u položky "Run As" zvolíme "Randoop Test Input" (viz. obr. 6.4). Jako položka



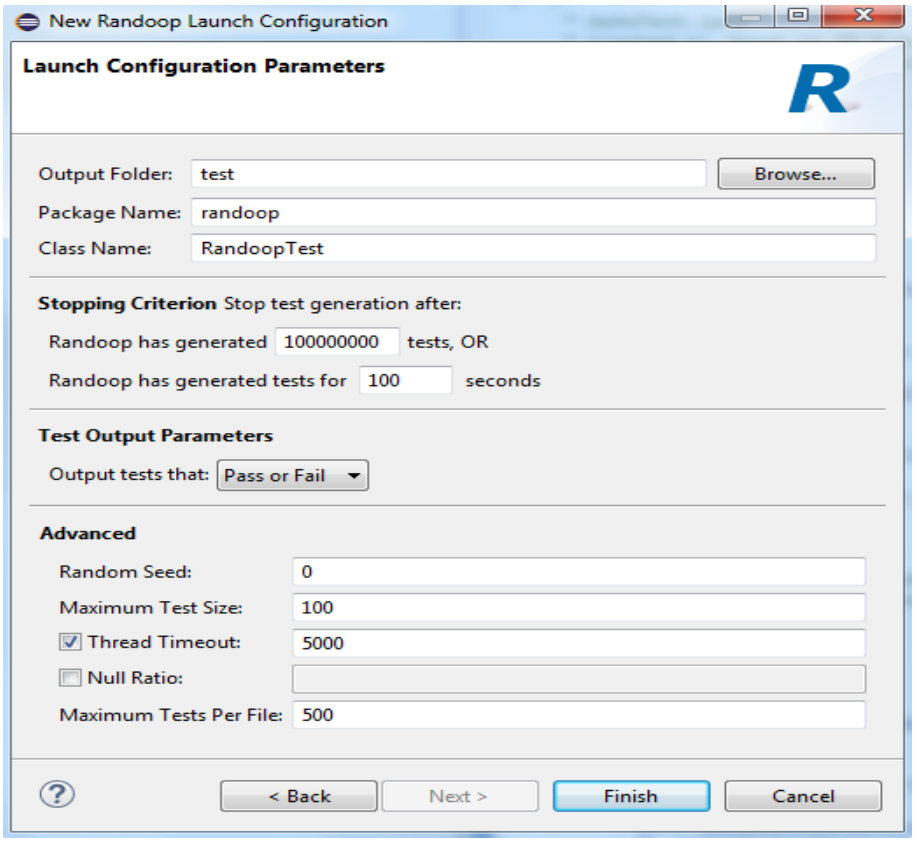
Obrázek 6.4: Generování testů nástrojem Randoop



Obrázek 6.5: Nastavení tříd k vygenerování testů u nástroje Randoop

testování může sloužit třída, balíček nebo klidně celý projekt. Po výběru se

objeví nové okno, kde je možné si zvolit, na které třídy se budou generovat testy (viz. obr. 6.5). Nyní máme dvě možnosti jak pokračovat dál. Můžeme rovnou nechat vygenerovat testy stisknutím "Finish" nebo přejít stisknutím "Next" do dalšího okna, kde si můžeme nastavit další parametry generování testů, jako například název výstupního adresáře, balíčku, třídy nebo po kolika vteřinách nebo testech má nástroj přestat testy generovat. Dále lze zvolit, jestli má nástroj generovat zdrojové kódy testů, které jsou úspěšné, neúspěšné nebo obě možnosti. Nastavit lze i maximální velikost testů, timeout vláken, jak často se má používat hodnota null při generování testů nebo počet testů na jeden zdrojový kód (viz. obr. 6.6).



Obrázek 6.6: Paramentry generování u nástroje Randoop

Po vygenerování testů se zdrojové kódy uloží do složky test, balíku randoop, s názvy `RandoopTest`, pokud nezměníme názvy v konfiguraci generování testů. Balíček obsahuje jeden soubor `RandoopTest.java`, který slouží ke spuštění všech souborů se samotnými testy. Tyto soubory jsou pojmenovány stejně, jen mají na konci názvu pořadové číslice.

Při použití spustitelného balíčku otevřeme příkazovou řádku a přesuneme se do složky, kde je spustitelný balíček spolu se zdrojovými kódy. Poté lze použít syntaxi `java -classpath randoop.jar randoop.main.Main <příkaz>`. Základními příkazy jsou `gentest` a `-testclass=Trida.java`. Výsledné zadání by pak vypadalo takto: `java -classpath randoop.jar randoop.main.Main gentests -testclass=Trida.java`. Nástroj následně vygeneruje testy do složky, kde se nachází spustitelný balíček. Celkový přehled příkazů lze najít na stránkách s manuálem k nástroji Randoop[1].

Listing 6.1: Test vytvořený nástrojem Randoop

```
public void test3() throws Throwable {
    if (debug) System.out.printf("%nRandoopTest0.test3");

    Auto.Auto var2 = new Auto.Auto(1.0d, 10.0d);
    double var3 = var2.getKolikNatankovat();
    Auto.Auto var6 = new Auto.Auto(100.0d, (-1.0d));
    var6.testPole();
    var6.testParse("");
    double var10 = var6.kolikUjedu();
    var6.setSpotreba(0.0d);
    var6.testPole();
    Auto.Auto var14 = var2.kopirujAuto(var6);
    var2.testPole();
    var2.testPole();
    double var17 = var2.getSpotreba();

    assertTrue(var3 == 1.0d);
    assertTrue(var10 == 0.0d);
    assertNotNull(var14);
    assertTrue(var17 == 10.0d);

  }
}
```

6.1.3 Spuštění vygenerovaných testů

Pro spuštění vygenerovaných testů ve vývojovém prostředí Eclipse je potřeba mít k projektu připojenou knihovnu JUnit, která je dostupná ke stažení na

[1] `https://randoop.github.io/randoop/manual/index.html`

stránkách `https://github.com/junit-team/junit4/wiki/Download-and-Install`. Knihovnu lze připojit tak, že klikneme pravým tlačítkem na projekt, zvolíme "Build Path" a "Add Libraries...". Poté zvolíme knihovnu JUnit a naimportujeme knihovnu do projektu. Dalším způsobem je stáhnout spustitelný balíček JUnit a ten přidat do classpath projektu. Když máme připojenou knihovnu, lze pustit vygenerované testy ve formátu JUnit.

6.1.4 Výsledky vygenerovaných testů

O výsledky testů se stará nástroj JUnit, protože Randoop testy umí pouze generovat. Pokud chceme zjistit pokrytí kódu, poslouží k tomuto účelu například nástroj EclEmma[2], který lze stáhnout jako plug-in do Eclipse.

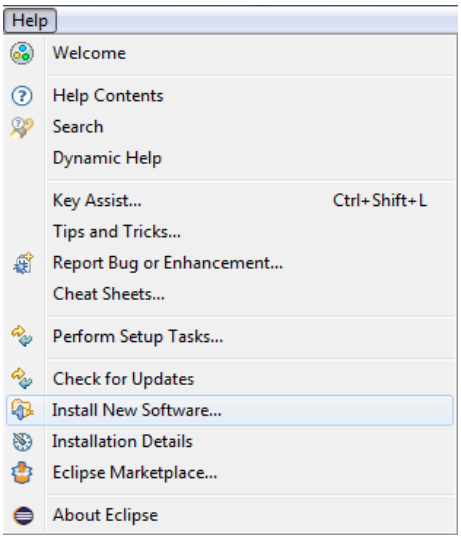
6.2 Ukázkový případ použití nástroje EvoSuite

Nástroj EvoSuite lze používat jako plug-in ve vývojovém prostředí Eclipse nebo ovládat samostatný balíček přes příkazovou řádku. Ukázka nástroje je doplněna o obrázky z vývojového prostředí při použití plug-inu a o popis základních příkazů při použití samostatného balíčku.

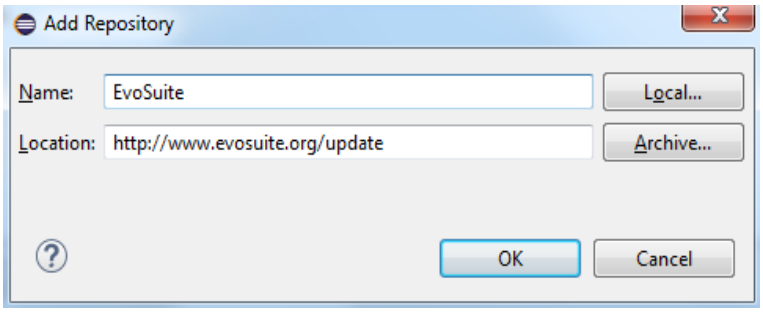
6.2.1 Instalace nástroje

Pokud chceme používat samostatný balíček, stačí na stránkách `http://www.evosuite.org/downloads/` stáhnout soubor `evosuite-1.0.3.jar` a můžeme nástroj spouštět a ovládat pomocí příkazové řádky. Když si vybereme plug-in (v tomto případě do vývojového prostředí Eclipse), je instalace nástroje jednoduchá. Po spuštění Eclipse vybereme v navigační liště položku "Help" a zvolíme "Install New Software..." (viz. obr. 6.7). Po kliknutí na "Add..." se zobrazí okno, kde se zadá název pod kterým se bude plug-in v nástroji zobrazovat a adresa `http://www.evosuite.org/update` (viz. obr. 6.8). Po potvrzení formuláře se zobrazí zaškrtávací seznam s nástrojem EvoSuite. Zaškrtnutím políčka s nástrojem a zvolením "Next >" (viz. obr. 6.9) nástroj

[2] `http://eclemma.org/index.html`



Obrázek 6.7: Instalace nástroje EvoSuite

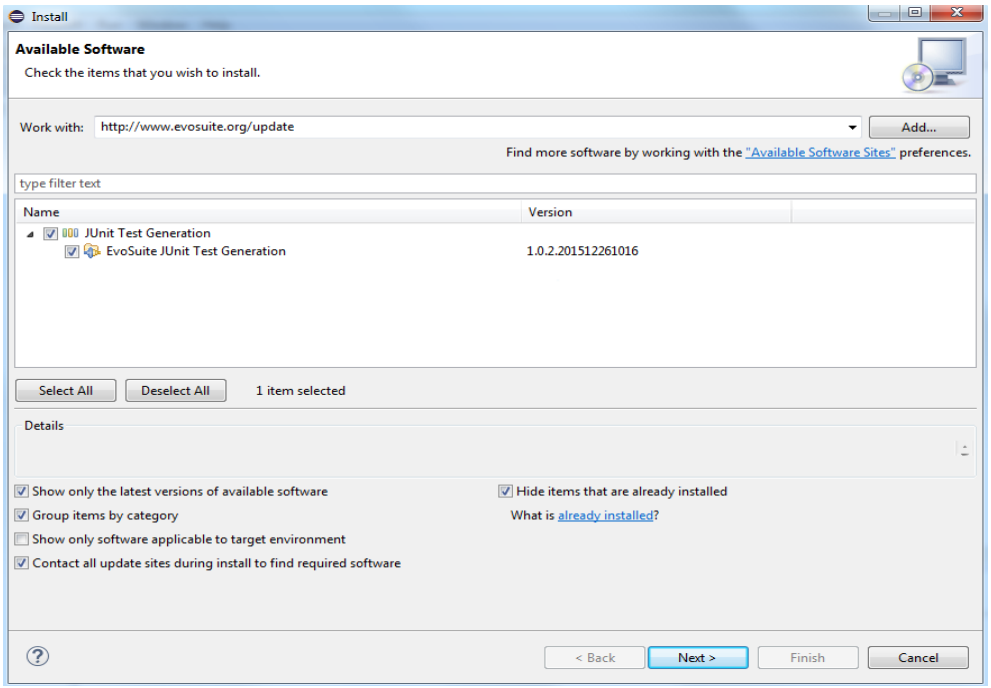


Obrázek 6.8: Instalace nástroje EvoSuite

nainstalujeme a po restartování vývojového prostředí je připraven k použití.

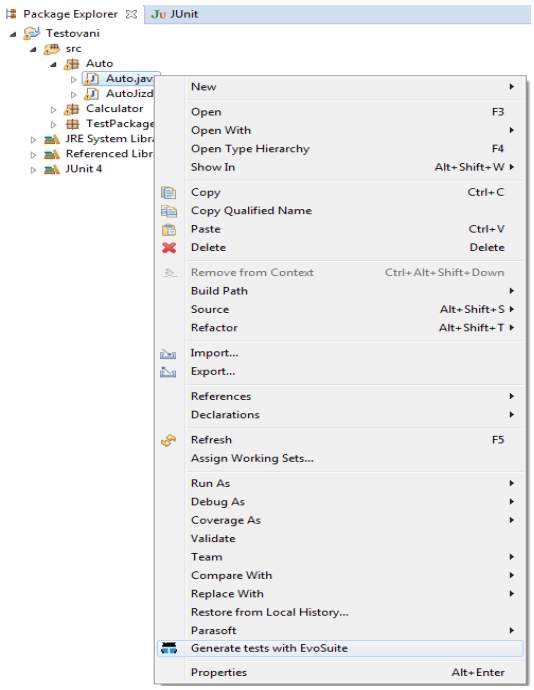
6.2.2 Generování testů

Když používáme EvoSuite jako plug-in, je generování testů velice jednoduché. Testy nelze generovat pro celé projekty nebo balíčky a tak v záložce "Package Explorer" stačí vybrat třídu, na kterou chceme vygenerovat testy, kliknout pravým tlačítkem a vybrat položku "Generate tests with EvoSuite" (viz. obr. 6.10). EvoSuite nabízí k nastavení několik parametrů generování testů. Do nastavení se lze dostat tak, že klikneme pravým tlačítkem na projekt a zvolíme "Properties". V zobrazeném okně pak vybereme záložku EvoSuite (viz.

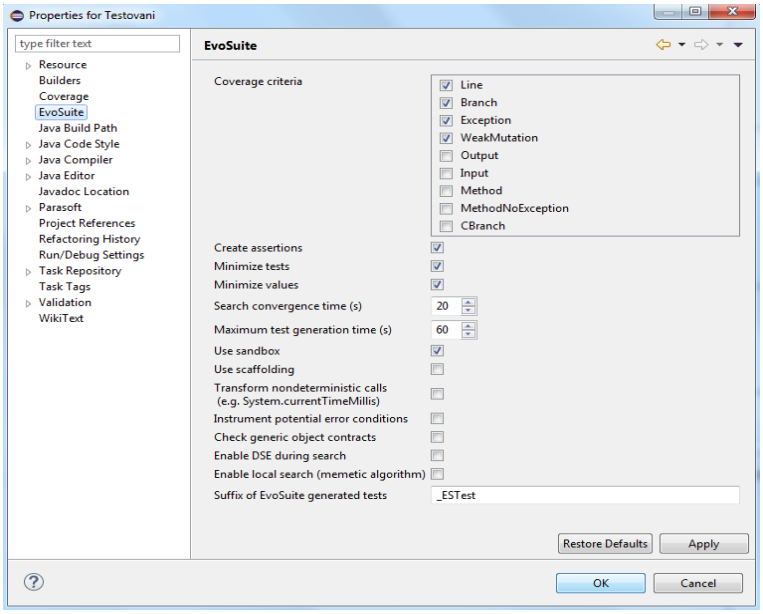


Obrázek 6.9: Instalace nástroje EvoSuite

obr. 6.11). První, co nástroj umožňuje, je nastavení kritérií pokrytí, na které se bude zaměřovat. Kritérií lze vybrat více, ale vždy minimálně jedno. Dále můžeme zvolit, jestli má nástroj minimalizovat testy a hodnoty, jak dlouho má nástroj testy generovat, jestli má nástroj generovat kontroly hodnot(tzv. asserts), jakou příponu mají mít vygenerované metody, aj.



Obrázek 6.10: Generování testů nástrojem EvoSuite



Obrázek 6.11: Paramentry generování u nástroje EvoSuite

Vygenerované testy se uloží do projektu do složky evosuite-tests. Názvy balíčků odpovídají projektu, ze kterého byly testy vygenerovány. U tříd to

platí stejně, jen se k nim dodá přípona nastavená v parametrech generování testů. Balíček s vygenerovanými testy obsahuje také jeden soubor, s koncovkou scaffolding, který obsahuje všechna potřebná nastavení ke spuštění testů.

Pokud se rozhodneme použít samostatný balíček, přesuneme se v příkazové řádce do složky s balíčkem. Pak lze zadat `java -jar evosuite.jar <příkaz>` a nejdůležitějším příkazem je `-projectCP <cesta k projektu>`. Přehled příkazů k vygenerování testů lze zobrazit zadáním `java -jar evosuite.jar -help`. Nástroj lze přes příkazovou řádku i nastavovat. Příkazy pro nastavení nástroje zobrazíme zadáním `java -jar evosuite.jar -listParameters`. Vygenerované testy uloží nástroj do složky evosuite-tests v místě, kde se nachází spustitelný balíček. Přes příkazovou řádku lze generovat testy i pro více tříd nebo i pro celý projekt. Na stránkách nástroje najdeme i základní popis k používání EvoSuite přes příkazovou řádku[3].

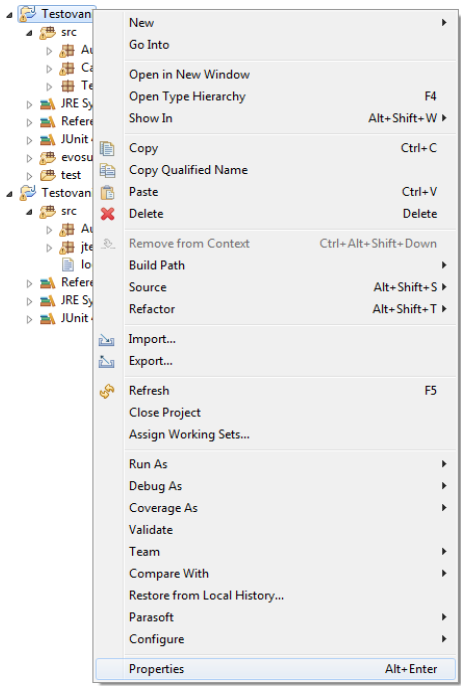
Listing 6.2: Test vytvořený nástrojem EvoSuite

```
public void test09() throws Throwable {
      Auto auto0 = new Auto(1.0, 1.0);
      Auto auto1 = new Auto((-2834.0), 1.0);
      Auto auto2 = auto0.kopirujAuto(auto1);
      assertEquals(1.0, auto2.getObjemNadrze(), 0.01D);
      assertEquals(0.0, auto2.getSpotreba(), 0.01D);
      assertEquals(0.0, auto1.getObjemNadrze(), 0.01D);
      assertNotNull(auto2);
}
```

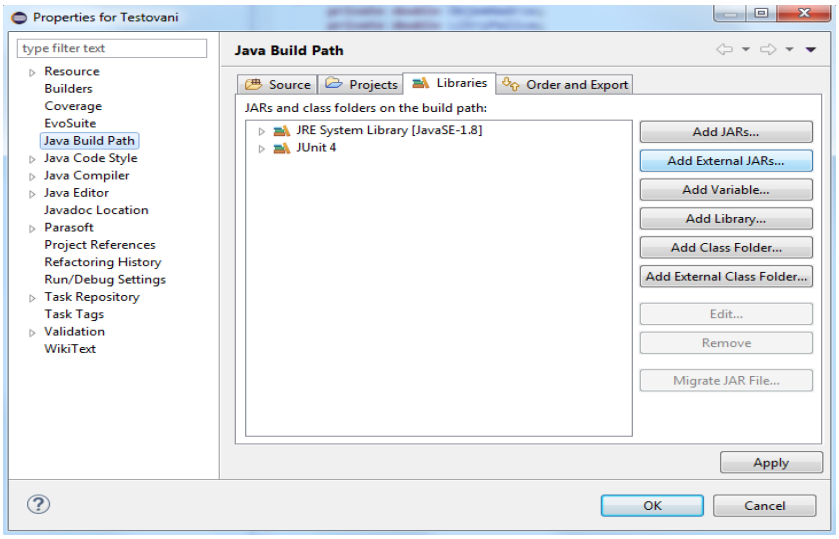
6.2.3 Spouštění vygenerovaných testů

Pro spuštění vygenerovaných testů ve vývojovém prostředí Eclipse potřebujeme mít k projektu připojenou knihovnu EvoSuite. Knihovna by se měla při instalaci plug-inu připojit sama. Pokud se tak nestane, je potřeba připojit knihovnu ručně. Nejdříve stáhneme EvoSuite balíček (viz. kapitola 6.2.1). Ve vývojovém prostředí klikneme pravým tlačítkem na projekt a vybereme položku "Properties" (viz. obr. 6.12). V nově zobrazeném okně vybereme "Java Build Path" a záložku "Libraries". Kliknutím na tlačítko "Add External JARs..." (viz. obr. 6.13) se zobrazí nové okno, kde najdeme a vybereme stáhnutý balíček, potvrdíme výběr balíčku i okno ve vývojovém prostředí a můžeme spouštět EvoSuite testy.

[3] `http://www.evosuite.org/documentation/commandline/`



Obrázek 6.12: Přidání knihovny EvoSuite



Obrázek 6.13: Přidání knihovny EvoSuite

6.2.4 Výsledky vygenerovaných testů

O výsledky testů se stará nástroj JUnit, protože EvoSuite testy pouze generuje. Pokud chceme zjistit pokrytí kódu testy, můžeme použít například nástroj EclEmma[4], který lze také stáhnout do Eclipse jako plug-in.

6.3 Ukázkový případ použití nástroje Jtest

Nástroj Jtest má svoje vlastní vývojové prostředí založené na platformě Eclipse anebo ho lze používat jako plug-in do vývojového prostředí Eclipse. Ukázka nástroje je doplněna obrázky z vývojového prostředí jak při použití plug-inu, tak svého vývojového prostředí. Jelikož je nástroj Jtest komerční, předchází jeho používání buď nákup nástroje nebo získání trial verze na omezenou dobu. K účelům této práce jsem se se společností, která Jtest vyvíjí, domluvil a získal od nich trial verzi na dobu čtrnácti dnů.

6.3.1 Instalace nástroje

Instalaci nástroje předcházelo ještě vytvoření účtu u společnosti Jtest. Po založení účtu pošle společnost žadateli odkaz na stránku se soubory ke stažení. Zde si vybereme, jestli chceme stáhnout plug-in do Eclipse anebo vývojové prostředí Jtest. Instalační soubory nejsou univerzální a jsou rozděleny podle operačních systémů, na kterém Jtest poběží. Po stažení námi požadované verze stačí spustit soubor, nástroj nainstalovat a Jtest je ihned připraven k použití.

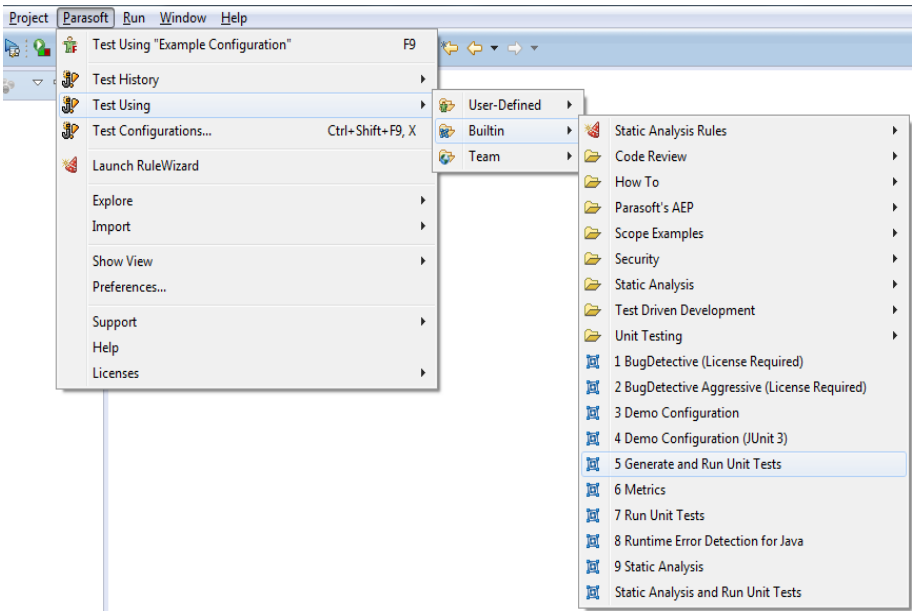
6.3.2 Generování testů

Jelikož je vývojové prostředí Jtest založeno na platformě Eclipse, probíhá generování testů stejně jak při použití plug-inu, tak při použití samostatného vývojového prostředí. Nástroj pracuje přes tzv. testovací konfigurace a pro generování unit testů už je v nástroji tato konfigurace přednastavena. Testy

[4] `http://eclemma.org/index.html`

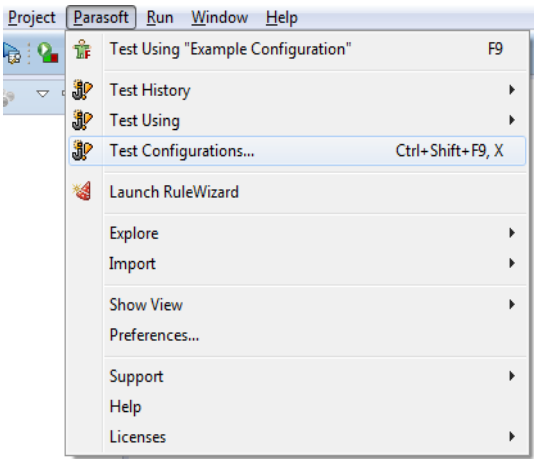
lze generovat pro jednu nebo více tříd, pro balíček nebo klidně pro celý projekt. Pro vygenerování testů tedy označíme pro jaké položky chceme testy generovat a v navigační liště zvolíme "Parasoft", pak položku "Test Using", dále "Builtin" a nakonec "Generate and Run Unit Tests" (viz. obr. 6.14). Druhou možností je kliknout na označenou položku v "Package Explorer" pravým tlačítkem a pokračovat stejně jako přes navigační lištu.

Pro nastavení parametrů generování je nutné upravit testovací konfi-

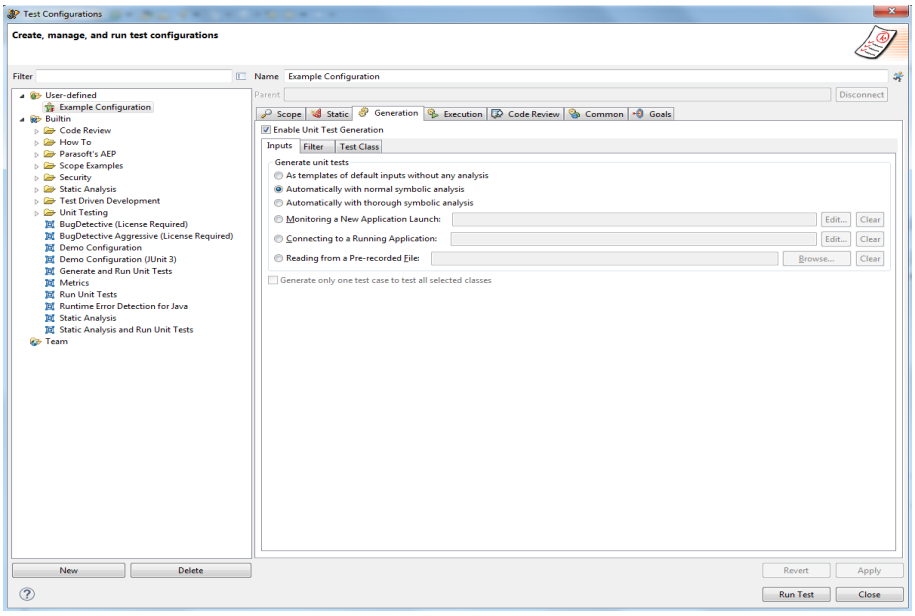


Obrázek 6.14: Generování testů nástrojem Jtest

guraci, kterou pro generování testů spouštíme. Přednastavené konfigurace nelze upravovat, ale nástroj dovoluje vytvořit vlastní konfigurace a ty upravit podle potřeb. Do nastavení konfigurací se dostaneme přes navigační lištu položkou "Parasoft" a zvolíme "Test Configurations..." (viz. obr. 6.15) V nově otevřeném okně se zobrazí seznam všech uložených konfigurací. Nové konfigurace můžeme přidávat do složky "User-defined". Všechny konfigurace v této složce je možné upravovat upravovat. Pro nastavení parametrů generování testů zvolíme konfiguraci, kterou chceme upravit, a zvolíme záložku "Generation" (viz. obr. 6.16). Nástroj nabízí různá nastavení. Nastavení parametrů je rozděleno do tří záložek. V záložce "Inputs" lze upravovat, co bude sloužit jako vstup pro generování testů (viz. obr. 6.16). V záložce "Filters" se nastavuje, jaké typy testů se budou generovat. Můžeme zvolit možnost, že se budou generovat jen testy, které zvýší určitý typ pokrytí, generování testů, které různým způsobem ošetřují výjimky nebo například generování testů pro různé typy přístupů k metodám (public, private, protected). Nástroj i

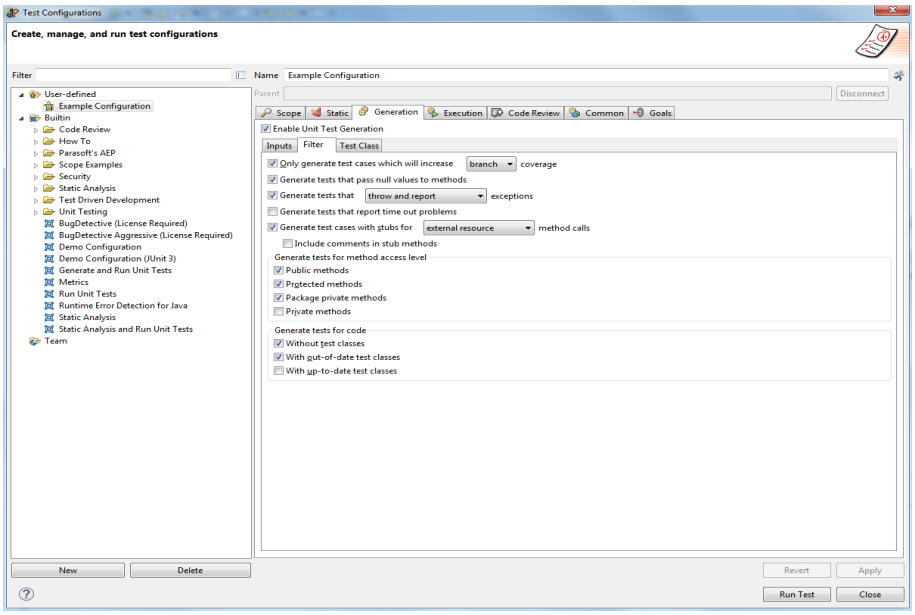


Obrázek 6.15: Parametry generování u nástroje Jtest

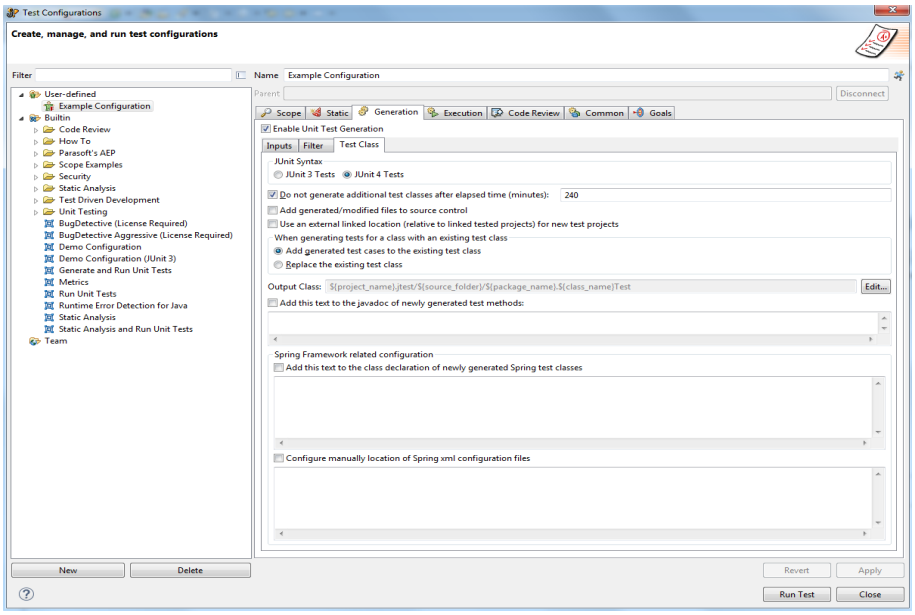


Obrázek 6.16: Parametry generování u nástroje Jtest

kontroluje, kdy byly testy vytvořeny, a proto lze nastavit, jestli se mají generovat testy pro kód, který má testy neaktuální, aktuální nebo ještě žádné nemá (viz. obr. 6.17). Poslední záložka "Test Class" obsahuje možnosti, jako jsou nastavení JUnit syntaxe, po jaké době se mají přestat generovat testy nebo kam se mají uložit vygenerované testy. Nástroj umožňuje i doplňovat už jednou vygenerované testy a pouze do nich dopsat nové testovací případy nebo testovací třídy kompletně nahradit novými (viz. obr. 6.18).



Obrázek 6.17: Parametry generování u nástroje Jtest



Obrázek 6.18: Parametry generování u nástroje Jtest

Vygenerované testy se v základním nastavení ukládají do nového projektu, který je strukturován stejně jako projekt, ze kterého se testy generovaly. Balíčky i třídy jsou pojmenovány stejně jako v testovaném projektu a k

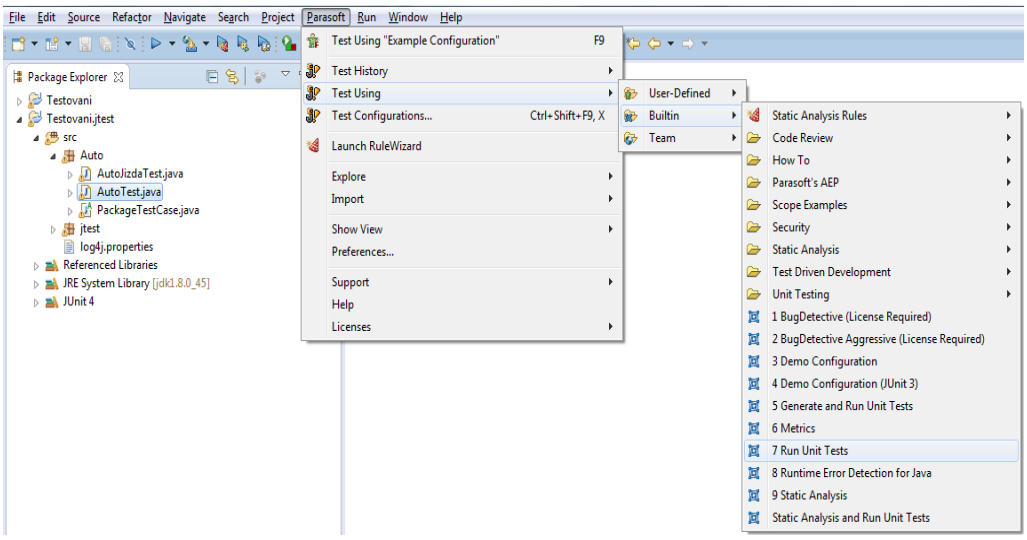
názvům tříd se doplní přípona `Test`. Balíček s vygenerovanými testovacími třídami obsahuje také jednu třídu, ve které se dá nastavit, co se má provést před spuštěním testů, po jejich ukončení, atd.

Listing 6.3: Test vytvořený nástrojem Jtest

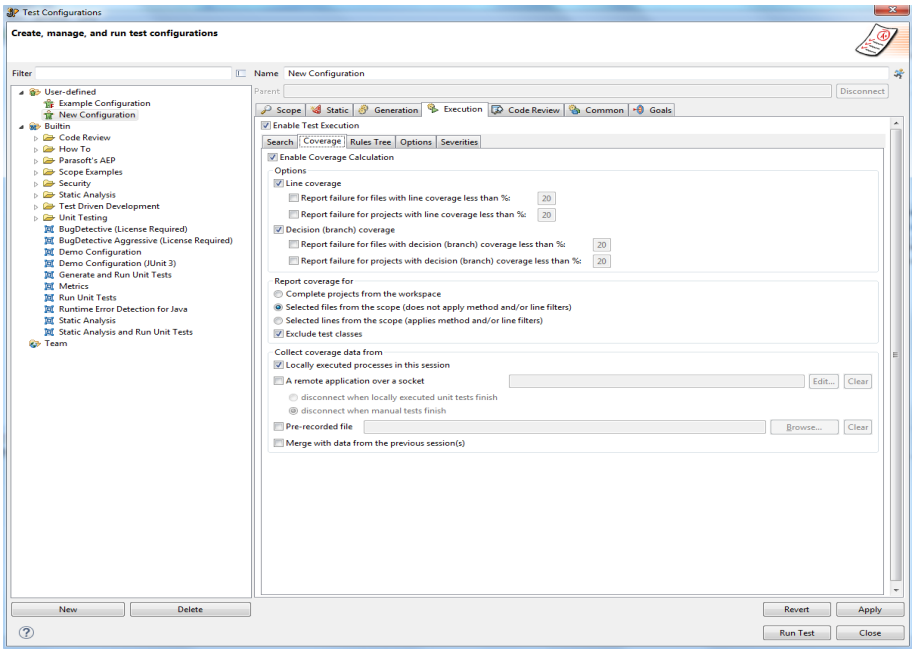
```
public void testAuto3() throws Throwable {
      Auto testedObject = new Auto(-1.0, 1.0);
      testedObject.setObjemNadrze(1.0);
      testedObject.setSpotreba(1.0);
      assertEquals(0.0, testedObject.getLitry(), 0.0);
      assertEquals(1.0, testedObject.getKolikNatankovat(), 0.0);
      assertEquals(1.0, testedObject.getObjemNadrze(), 0.0);
      assertEquals(1.0, testedObject.getSpotreba(), 0.0)
}
```

6.3.3 Spouštění vygenerovaných testů

Spouštění testů v nástroji Jtest probíhá použitím testovací konfigurace. Postup použití je stejný jako u generování testů. Označíme testy, které chceme spustit, a buď klikneme pravým tlačítkem a vybereme položku "Parasoft", nebo vybereme po označení položku "Parasoft" v navigační liště. Dále pokračujeme přes "Test Using", "Builtin" a "Run Unit Tests" (viz. obr. 6.19).



Obrázek 6.19: Spouštění testů u nástroje Jtest

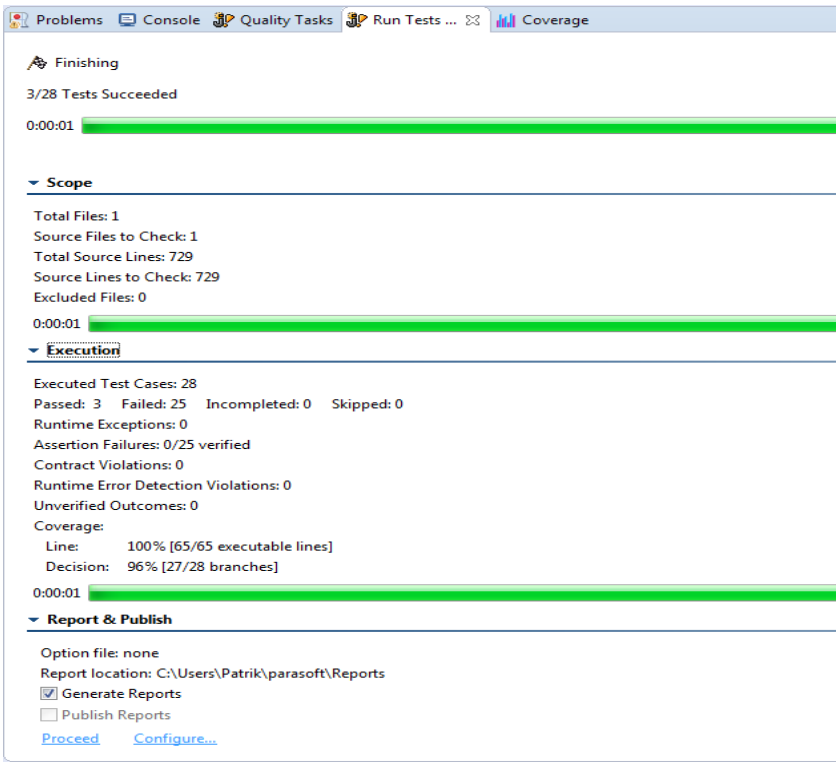


Obrázek 6.20: Parametry spouštění testů u nástroje Jtest

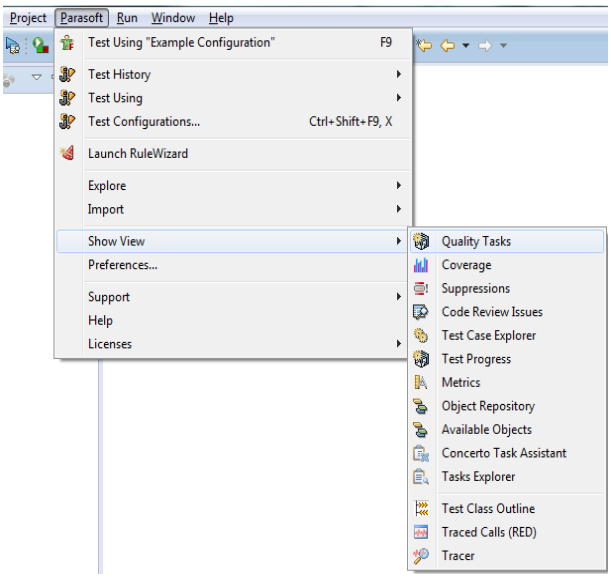
Jelikož se testy spouští přes testovací konfiguraci, lze si nastavit, jakým způsobem se budou testy spouštět. V nastavení konfigurací zvolíme konfiguraci, kterou chceme upravit, a zvolíme záložku ”Execution” (viz. obr. 6.20). Jsou zde různé možnosti nastavení, jako například jaké testy se budou spouštět, podle zadaného vzorce názvů, jaké typy pokrytí se bude kontrolovat, jaké chyby podle závažnosti se budou zapisovat do výsledného reportu a mnoho dalších možností.

6.3.4 Výsledky vygenerovaných testů

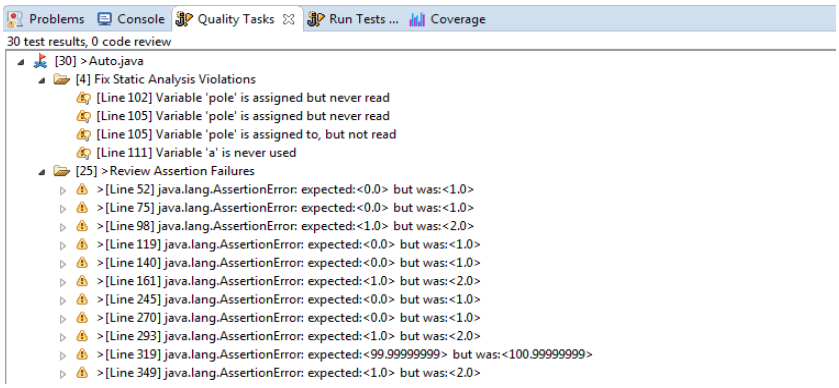
O výsledky testů se stará samotný nástroj Jtest. Po spuštění testů se otevře nová záložka ve vývojovém prostředí s výsledky testů. Vypíše se například, kolik testů bylo spuštěno, které byly úspěšné nebo neúspěšné, jaké bylo pokrytí testy, atd. (viz. obr. 6.21). Pokud testy byly neúspěšné, je dobré vidět, které testy selhaly a kde. Na to slouží záložka ”Quality Tasks”. Tu zobrazíme přes navigační lišty v položce ”Parasoft”, dále ”Show View” a ”Quality Tasks” (viz. obr. 6.22) V této záložce se zobrazí kde v kódu odhalily testy nesrovnalosti a jaké (viz. obr. 6.23). Dvojím kliknutím levým tlačítkem na nalezený problém se vývojové prostředí přesune přímo do kódu na danou



Obrázek 6.21: Výsledky průběhu testů

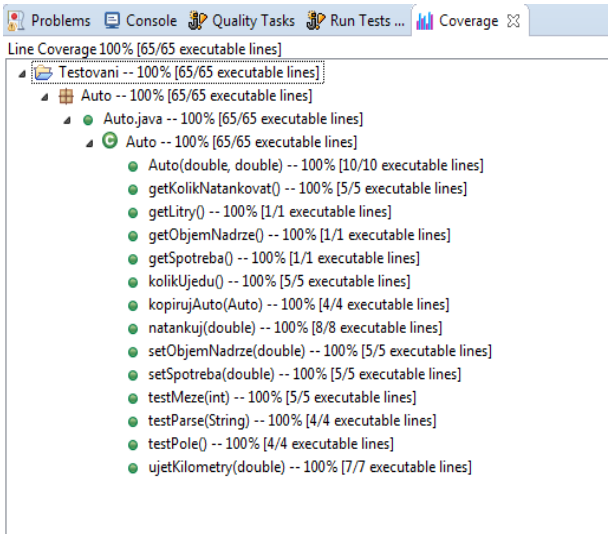


Obrázek 6.22: Otevření záložky Quality Tasks



Obrázek 6.23: Odhalené nesrovnalosti při testování

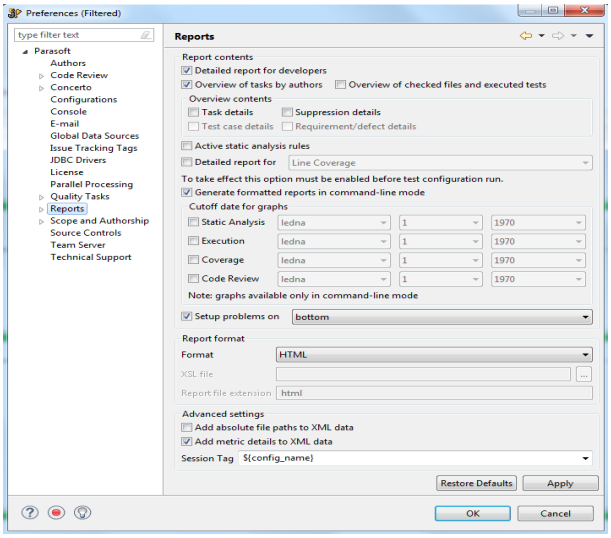
řádku. Nástroj Jtest měří i pokrytí kódu a nemusí se pro tento účel používat žádné externí nástroje. Pokud chceme zobrazit pokrytí kódu, tak v navigační liště v položce ”Parasoft” zvolíme ”Show View” a ”Coverage” (viz. obr. 6.22). Otevře se nová záložka, kde je vidět pokrytí pro testované třídy (viz. obr. 6.24).



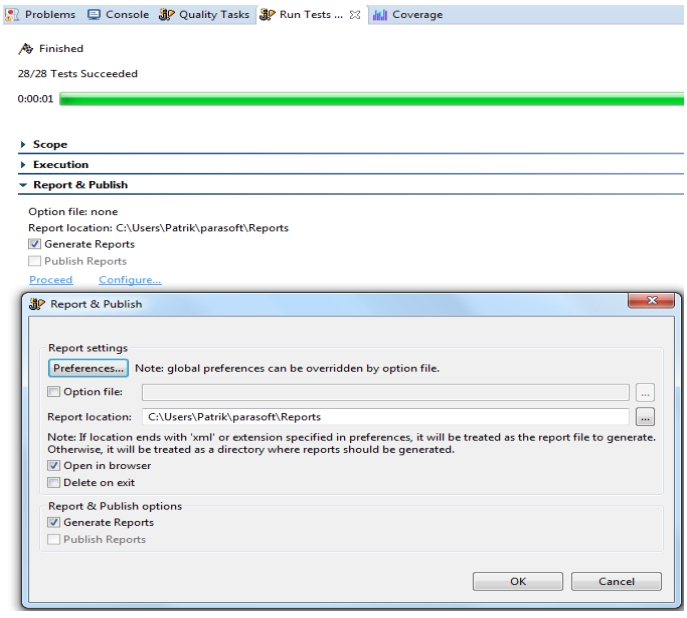
Obrázek 6.24: Pokrytí kódu

Nástroj dále umožňuje vygenerovat report o průběhu testu. Report se dá generovat v různých formátech jako například hmtl, pdf, xml, aj. Do nastavení reportů se dostaneme přes navigační lištu v položce ”Parasoft”, ”Preferences...” a po zobrazení nového okna v ”Reports” (viz. obr. 6.25). Reporty lze generovat pouze po spouštění testů, kde v záložce s výsledky, v sekci

"Report & Publish", klikneme na "Proceed". Umístění reportu je v této sekci zobrazeno také a lze ho nastavit po kliknutí na "Configure" (viz. obr. 6.26). Pokud jsou při testech nalezeny chyby, vygeneruje se i report pro vývojáře s přehledem chyb.



Obrázek 6.25: Nastavení reportů



Obrázek 6.26: Nastavení reportů

7 Vyhodnocení nástrojů

V této kapitole otestujeme vybrané nástroje a zhodnotíme jejich výsledky. Nástroje budu testovat na mnou vytvořené aplikaci. Jedná se o jednoduchou třídu, která má simulovat objekt auta. Konstruktor objektu obsahuje nastavení spotřeby auta a objemu nádrže. Metody na ovládání auta jsou potom metody jako `natankuj(double litry)`, kde se přidávají litry paliva do nádrže až do jejího maximálního objemu, `ujetKilometry(double kilometry)`, kdy metoda ubírá palivo z nádrže podle nastavené spotřeby auta, `kolikUjedu()`, abychom mohli zjistit jak daleko můžeme s autem dojet, atd. Třída poté obsahuje ještě čtyři metody, které s objektem auta nesouvisí, ale slouží k zanesení dalších chyb do programu.

Dále je v kapitole popsáno použití jednotlivých nástrojů, konfigurace jejich spuštění, výsledky generování testů a hardware a operační systém, na kterém jsem nástroje testoval. U výsledků generování se kontroluje například, kolik vygeneroval nástroj testů, kolik bylo celkových řádků kódu u vygenerovaných testů, jaké bylo pokrytí testy, jak dlouho se testy generovaly a jaké chyby nástroje odchytily. Na konci kapitoly je srovnání nástrojů ve dvou tabulkách, kde v jedné jsou údaje o vygenerovaných testech a ve druhé přehled nalezených chyb.

7.1 Zanesené chyby

V programu je několik typů chyb. Cílem je zjistit, jestli nástroje chyby odhalí nebo nikoli. Zanesené chyby jsou:

- Vrácení null z metody místo očekávaného objektu - v metodě `kopirujAuto(Auto auto)`, která má vrátit objekt nově vytvořeného auta, vrátím null
- Vyhození výjimky před dokončením metody - v metodě `testPole()` je zanesena chyba překročení indexu pole
- Vstupní hodnota parametru mimo očekávané meze - v metodě `test-`

`Meze(int a)`, nastavím po zavolání metody hodnotu `a` tak, aby nevyhovovala podmínce

- Vrácení nesprávné hodnoty - u metod, které vracejí číselnou hodnotu, navrátím pevně zadanou hodnotu a ne vypočítanou
- Přepsání vnitřku metody - přepíši vnitřek metody, aby přenastavila jiné proměnné než původně měla

7.2 Použití nástrojů

Nástroje byly používány ve vývojovém prostředí Eclipse. Randoop a Evosuite lze ovládat i přes příkazovou řádku, ale ani jeden z nástrojů se mi nepodařilo takto použít. Všechny příklady generování a použití nástrojů probíhaly na operačním systému 64-bitové verzi Windows 7, čtyř jádrovém procesoru Intel Core i5-2400 s frekvencí 3.10 GHz a nainstalovaná operační paměť byla 8GB.

U všech nástrojů vygeneruji testy s použitím různých konfigurací. Nejdříve v základním nastavení a poté upravím parametry generování u všech nástrojů. V přiloženém CD jsou zdrojové soubory. Jeden obsahuje aplikaci bez chyb a další obsahují zanesené chyby. Postup je takový, že vytvoříme nový projekt v Eclipse a v adresáři src balíček Auto. Do této třídy následně zkopírujeme zdrojový soubor třídy, která je bez chyb. Necháme nástroj vygenerovat testy a třídu Auto nahradíme za soubor s chybami. Následně testy pustíme a budeme kontrolovat jaké chyby odhalí.

7.2.1 Použití nástroje Randoop

Jako první konfiguraci pro vygenerování testů použiji základní, kterou má Randoop nastavenou. Testy vygenerujeme podle návodu v kapitole 6.1.2. Nástroj generoval testy celých 100 vteřin, které byly nastaveny v základní konfiguraci. Randoop zobrazi vygeneroval 5 491 testů a ty rozdělil do 11 testovacích tříd. V těchto jedenácti třídách nástroj vygeneroval 274 286 řádků kódu. Vzhledem k tomu, že testovaná třída má pouze 13 metod, je toto číslo poměrně velké. Tato sada testů měla 100% pokrytí řádek kódu a 96,2% pokrytí větví.

Ve druhé konfiguraci jsem zadal, aby nástroj přestal generovat testy po 1000 testech anebo se zastavil po 60 vteřinách. Nastavil jsem také poměr použití null na 0,5. Nástroj vygeneroval 473 testů, generování trvalo 5 vteřin a

bylo vytvořeno 16 513 řádků kódu. Pokrytí řádek bylo 93,4% a pokrytí větví 92,3%.

V poslední konfiguraci jsem zvolil generování (podle mě úměrné) k velikosti aplikace. Nastavil jsem proto generování pouhých 80 testů a poměr využití null na 1. V tomto případě generování trvalo 1 vteřinu a vytvořilo se 33 testů na 672 řádcích kódu. Pokrytí kódu bylo znatelně horší. Při takto malé sadě testů bylo pokrytí řádek 65,5% a pokrytí větví pouhých 46,2%.

Při první konfiguraci odhalil nástroj všechny chyby. Tento výsledek byl očekávaný díky 5 491 testům. Druhá konfigurace už tak přesná nebyla. Nepodařilo se jí zjistit přepsání vnitřku metody a vrácení jiné než očekávané hodnoty. Poslední konfigurace odhalila pouze jednu zanesenou chybu, a to vyhození výjimky. Dalo se předpokládat, že tato konfigurace neodhalí vše, už z toho důvodu, že mají testy velmi nízké pokrytí kódu.

7.2.2 Použití nástroje EvoSuite

Nástroj EvoSuite umožňuje generovat testy pro jedno či více pokrytí a lze i minimalizovat testovací případy a hodnoty. Počet konfigurací, které by byly vhodné, je zde více. Já zvolil konfigurace tři. Testy jsem generoval podle návodu z kapitoly 6.2.2.

V první konfiguraci jsem nechal nástroj v základním nastavení. Nástroj generoval testy pro pokrytí řádek i větví a měl nataveno minimalizování testů a hodnot. Testy se generovaly 40 vteřin a vytvořilo se 25 testů na 231 řádcích. Pokrytí řádek i větví bylo 100%.

V druhé konfiguraci jsem zakázal minimalizovat testovací případy a hodnoty a nastavil jsem generování na pokrytí řádek. Za 15 vteřin vygeneroval nástroj 4 testy a 127 řádků kódu. I tady bylo pokrytí řádek a větví 100%.

Třetí konfigurace měla také zakázáno minimalizovat testovací případy a hodnoty a byla zaměřena na pokrytí větví. Nástroj vygeneroval 8 testů, 147 řádků kódu a testy vytvářel 13 vteřin. Pokrytí řádek a větví bylo stejné jako v ostatních konfiguracích a to 100%.

První konfigurace, i přes menší počet vygenerovaných testů, odhalila všechny zanesené chyby. Druhá konfigurace nedokázala odhalit změnu meze a přepsání vnitřku metody. Tento výsledek by se dal připsat nastavení zakázání minimalizace, kdy nástroj nerozdělil testy do menších celků. Třetí konfigurace neodhalila přepsání vnitřku metody.

7.2.3 Použití nástroje Jtest

Jtest nenabízí mnoho konfigurací, které by se pro účely této práce značně lišily. Použiji proto dvě konfigurace. První konfigurace bude generovat testy, které zvětší pokrytí řádek, a druhá konfigurace bude nastavena, aby generovala testy, které zvětší pokrytí větví. Testy jsem vygeneroval podle návodu v kapitole 6.3.2.

U první konfigurace, kdy nástroj generoval testy zaměřené na pokrytí řádek, trvalo generování 3 vteřiny. Vytvořilo se 12 testů a 367 řádků kódu. Porytí řádek bylo 97% a pokrytí větví 85%.

V druhé konfiguraci se generovaly testy zaměřené na pokrytí větví. Za 5 vteřin vygeneroval nástroj 25 testů na 663 řádcích kódu. Pokrytí řádek kódu bylo 98% a pokrytí větví 96%. Je vidět, že různá nastavení pokrytí hrála roli i u takto malé aplikace. Rozdíl mezi konfiguracemi u pokrytí větví byl 11%.

I přes nízký počet testů se oběma konfiguracím podařilo odhalit většinu chyb. Neporadily si ovšem se změnou vrácené hodnoty.

7.3 Srovnání nástrojů

Nástroj Randoop dokázal odhalit v jedné konfiguraci všechny chyby, ale potřeboval k tomu více jak 5 000 testů. Ostatní konfigurace už nebyly tak přesné. Tento výsledek připisuji metodě generování, kterou Randoop používá. Jelikož generuje testy náhodně, je větší šance, že se mu povede pokrýt větší stavový prostor, pokud vygeneruje více testů. Proto také další dvě konfigurace neodhalily všechny chyby. Při srovnatelném počtu testů nedokázal Randoop dosáhnout tak vysokého pokrytí kódu jako ostatní nástroje.

Nástroj EvoSuite mě překvapil v tom, že dokázal odhalit všechny chyby v aplikaci a potřeboval k tomu jen 25 testů. Další dvě konfigurace už tak úspěšné nebyly. Důležitý faktor v tomto případě hrála minimalizace testů a hodnot. Pokud nástroji povolíme minimalizaci, rozdělí testy na menší celky, testů vygeneruje více a je schopen odhalit více chyb. Testy jsou také přehlednější. Při vypnuté minimalizaci může mít jeden test i přes 30 řádek a je těžší se v nich zorientovat. Nástroj také jako jediný dokázal vygenerovat testy se 100% pokrytím řádek a větví.

Nástroj Jtest neodhalil všechny chyby ani v jedné konfiguraci. Jtest neměl takové možnosti nastavení generování testů jako ostatní nástroje. I když měli konfigurace stejný výsledek při odhalování chyb, používal bych spíše

generování na pokrytí větví. Nástroj sice vygeneroval více jak dvojnásobek testů, ale dosáhl většího pokrytí větví. Pokrytí řádek bylo srovnatelné v obou konfiguracích. Oproti ostatním nástrojům neslouží Jtest pouze ke generování testů. Jako komerční nástroj je vyvíjen pro více účelů a nástroj lze použít ve všech fázích vývoje software.

Nástroj Randoop bych použil spíše na zkoušku stavového prostoru. Díky náhodnému generování je možnost, že se nástroj trefí do vstupních hodnot, které by ostatní nástroje nevyzkoušely. Nástroj bych ovšem nedoporučil k častějšímu použití, protože potřebuje vygenerovat velké množství testů, aby byl efektivní. EvoSuite se osvědčil jak v odhalování chyb, tak v pokrytí kódu. Oproti nástroji Randoop vygeneroval jen zlomek testů a dosáhl stejného výsledku. EvoSuite je podle mě dostačující pro testování software, pokud nejsme ochotni platit za komerční nástroje, ať už se jedná o menší nebo větší projekt. Jtest sice nebyl v odhalování chyb dokonalý, ale nástroj podporuje mnoho dalších funkčností, které jsou použitelné při vývoji software. Nástroj bych nepoužil na menší projekty už z toho důvodu, že je placený. Při větších projektech by už dávalo smysl nástroj použít, hlavně díky podpoře týmové spolupráce.

Nástroje	**Konfigurace**	**Doba běhu**	**Testů**	**Řádek**	**Pokrytí řádek**	**Pokrytí větví**
Randoop	10000 t/100 s/null 0	100 s	5 491	274 286	100%	96,2%
	1000 t/60 s/ null 0,5	5 s	473	16 513	93,4%	92,3%
	80 t/100 s/ null 1	1 s	33	672	65,5%	46,2%
EvoSuite	Pokrytí obě, Min. zap.	40 s	25	231	100%	100%
	Pokrytí řádek, Min. vyp.	15 s	4	127	100%	100%
	Pokrytí větví, Min. vyp.	13 s	8	147	100%	100%
Jtest	Pokrytí řádek	3 s	12	367	97%	85%
	Pokrytí větví	5 s	25	663	98%	96%

Tabulka 7.1: Srovnání nástrojů při generování testů

Nástroje	**Konfigurace**	**Null**	**Výjimka**	**Mimo mez**	**Hodnota**	**Vnitřek metody**
Randoop	10000 t/100 s/null 0	✓	✓	✓	✓	✓
	1000 t/60 s/ null 0,5	✓	✓	✓	✗	✗
	80 t/100 s/ null 1	✗	✓	✗	✗	✗
EvoSuite	Pokrytí obě, Min. zap.	✓	✓	✓	✓	✓
	Pokrytí řádek, Min. vyp.	✓	✓	✗	✓	✗
	Pokrytí větví, Min. vyp.	✓	✓	✓	✓	✗
Jtest	Pokrytí řádek	✓	✓	✓	✗	✓
	Pokrytí větví	✓	✓	✓	✗	✓

Tabulka 7.2: Srovnání nástrojů při odhalování chyb

8 Závěr

Tématem této práce byla analýza využití metod testování software a jejím cílem bylo popsat různé metody generování testů, prozkoumat trh s nástroji, které umí automaticky generovat testy a porovnat vybrané nástroje.

V teoretické části práce jsou popsány typy různých testů. Testy jsou rozděleny do dvou základních skupin a to funkční testy a nefunkční testy. U každé skupiny jsou pak vypsány jednotlivé typy testů s popisem, jak testy fungují a na co se používají.

V další části práce jsem prohledal trh s nástroji na generování testů. Při hledání jsem bral ohled na několik kritérií a pak jsem vytvořil přehled několika nástrojů. Zajímal jsem se především o to, jestli nástroje umí přímo generovat testy, ale také jaká je cena nástroje, jaká je dostupnost nástroje, jaké technologie nástroj podporuje a v neposlední řadě jestli je nástroj dále vyvíjen, anebo se jedná o mrtvý projekt. Nakonec jsem vybral tři nástroje, které uměly automaticky vygenerovat testy a každý používal jinou metodu generování testů. Byly to nástroje Randoop, EvoSuite a Jtest.

K těmto nástrojům jsem sestrojil ukázkové případy použití, ve kterých jsem popsal instalaci nástroje, nastavení nástroje, generování testů, spuštění testů a kontrolu výsledků. Ukázkové případy jsem psal tak, aby je mohl použít i někdo, kdo se s nástroji nikdy nesetkal. Popisy ukázkových případů jsou doplněny o obrázky pro snazší orientaci při ovládání nástrojů.

V poslední části práce jsem vytvořil jednoduchou aplikaci a nechal nástroje vygenerovat testy v několika různých konfiguracích. Do aplikace jsem následně zanesl chyby a zkoumal jsem, jak nástroje obstojí v hledání chyb. Všechny chyby odhalily nástroje Randoop a EvoSuite. Pokud bych se měl rozhodovat mezi těmito dvěma nástroji, doporučil bych EvoSuite, protože vytvořil menší množinu testů s větším pokrytím oproti nástroji Randoop a je zdarma. Jtest sice neodhalil jednu chybu v kódu, ale nástroj je plně využitelný ve všech fázích vývoje software a ne jen pro generování testů. Nedoporučil bych ovšem jeho používání na malé projekty, protože je placený a mnoho jeho funkcí by pravděpodobně zůstalo nevyužito.

Jako přínos této práce bych označil shrnutí typů testů a metod generování testů, vytvoření ukázkových případů a reálné použití nástrojů spolu s vyhodnocením výsledků.

Seznam tabulek

3.1 Údaje o nástroji Conformiq Designer 7
3.2 Údaje o nástroji Test Studio 8
3.3 Údaje o nástroji Jubula 9
3.4 Údaje o nástroji Agitar One 9
3.5 Údaje o nástroji Randoop 10
3.6 Údaje o nástroji Concordion 11
3.7 Údaje o nástroji Jtest 12
3.8 Údaje o nástroji EvoSuite 12
3.9 Údaje o nástroji SoapUI 13

4.1 Srovnání metod generování testů 20

5.1 Kritéria popisu nástrojů 21

7.1 Srovnání nástrojů při generování testů 53
7.2 Srovnání nástrojů při odhalování chyb 54

Literatura

[1] JORGENSEN, Paul. Software testing. Boca Raton: Auerbach. 2008. 416 s. ISBN 0-8493-7475-8

[2] NEUVEDEN. Funkční a nefunkční testy [online]. [cit. 19.2.2016]. Dostupný na WWW: http://testovanisoftwaru.cz/tag/nefunkcni-testy/

[3] NEUVEDEN. Unit testing [online]. [cit. 26.2.2016]. Dostupný na WWW: https://cs.wikipedia.org/wiki/Unit_testing

[4] NEUVEDEN. Druhy testování v procesu vývoje SW [online]. [cit. 26.2.2016]. Dostupný na WWW: http://www.swtestovani.cz/index.php/uvod-do-testovani/18-druhy-testovani-v-procesu-vyvoje-sw

[5] NEUVEDEN. Fáze a úrovně provádění testů [online]. [cit. 26.2.2016]. Dostupný na WWW: http://testovanisoftwaru.cz/category/metodika-testovani/druhy-typy-a-kategorie-testu/

[6] NEUVEDEN. Zátěžové testy software [online]. [cit. 26.2.2016]. Dostupný na WWW: http://testovanisoftwaru.cz/testing/zatezove-testy-software/

[7] NEUVEDEN. Penetrační test [online]. [cit. 26.2.2016]. Dostupný na WWW: https://cs.wikipedia.org/wiki/Penetra%C4%8Dn%C3%AD_test

[8] NEUVEDEN. Uživatelské testování [online]. [cit. 26.2.2016]. Dostupný na WWW: https://cs.wikipedia.org/wiki/U%C5%BEivatelsk%C3%A9_testov%C3%A1n%C3%AD

[9] VEFIRYSOFT. Conformiq User Manual [online]. [cit. 9.3.2016]. Dostupný na WWW: http://www.verifysoft.com/ConformiqManual.pdf

[10] TELERIK. Test Studio Documentation [online]. [cit. 9.3.2016]. Dostupný na WWW: http://docs.telerik.com/teststudio/general-information/for-developers

[11] AGITAR TECHNOLOGIES. Agitar One [online]. [cit. 9.3.2016]. Dostupný na WWW: http://www.agitar.com/solutions/products/agitarone.html

[12] RANDOOP. Randoop [online]. [cit. 9.3.2016]. Dostupný na WWW: https://randoop.github.io/randoop/

[13] CONCORDION. Concordion [online]. [cit. 9.3.2016]. Dostupný na WWW: http://concordion.org

[14] PARASOFT. Unit Testing [online]. [cit. 10.3.2016]. Dostupný na WWW: https://www.parasoft.com/capability/unit-testing/

[15] EVOSUITE. About EvoSuite [online]. [cit. 10.3.2016]. Dostupný na WWW: http://www.evosuite.org/evosuite/

[16] MATHUR, Aditya P. Foundation of Software Testing. Indie: Pearson, 2014, ISBN 978-93-325-2157-5.

[17] NEUVEDEN. Funkční vs nefunkční testování [online]. [cit. 31.3.2016]. Dostupný na WWW: http://www.swtestovani.cz/index.php/uvod-do-testovani/22-funkni-vs-nefunkni-testovani

[18] PACHECO, Carlos; LAHIRI, Shuvendu K.; ERNST, Michael D.;BALL, Thomas. In ICSE 2007: Proceedings of the 29th International Conference on Software Engineering [online]. Minneapolis, MN, USA, 2007 [cit. 31.3.2016]. Dostupný na WWW: http://homes.cs.washington.edu/~mernst/pubs/feedback-testgen-icse2007.pdf

[19] SWAIN, Ranjita; PANTHI, Vikan; BENEHRA, Prafulla Kumar; MOHAPATRA, Durga Prasad. International Journal of Computer Applications(0975-8887), Volume 42-No.7, March 2012 [online]. [cit. 31.3.2016]. Dostupný na WWW: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.259.1439&rep=rep1&type=pdf

[20] SAMUEL, Philip; MALL, Rajib; KANTH, Pratyush. Automatic test case generation from UML communication diagrams. Information and Software Technology 49 (2007) 158–171 [online]. [cit. 1.4.2016]. Dostupný na WWW: http://www.sciencedirect.com/science/article/pii/S0950584906000474

[21] NEUVEDEN. Predikát [online]. [cit. 1.4.2016]. Dostupný na WWW: https://cs.wikipedia.org/wiki/Predik%C3%A1t_%28logika%29

[22] KUNDU, Debasish; SAMANTA Debasis. A Novel Approach to Generate Test Cases from UML Activity Diagrams. Journal of Object Technology, Volume 8-No.3, May-June 2009 [online]. [cit. 1.4.2016]. Dostupný na WWW: http://www.jot.fm/issues/issue_2009_05/article1.pdf

[23] NEUVEDEN. Diagram aktivit [online]. [cit. 1.4.2016]. Dostupný na WWW: http://mpavus.wz.cz/uml/uml-b-activity-3-2-3.php

[24] GOMES, Ivo; MORGADO, Pedro; GOMES, Tiago. An overview on the Static Code Analysis approach in Software Development [online]. [cit. 3.4.2016]. Dostupný na WWW: http://paginas.fe.up.pt/~ei05021/TQSO%20-%20An%20overview%20on%20the%20Static%20Code%20Analysis%20approach%20in%20Software%20Development.pdf

[25] FRASER, Gordon; ARCURI, Andrea. Automated Test Generation for Java Generics [online]. [cit. 3.4.2016]. Dostupný na WWW: http://www.evosuite.org/wp-content/papercite-data/pdf/swqd14_generics.pdf

[26] GALEOTTI, Juan Pablo; FRASER, Gordon; ARCURI, Andrea; Improving Search-based Test Suite Generation with Dynamic Symbolic Execution [online] [cit. 5.4.2016]. Dostupný na WWW: http://www.evosuite.org/wp-content/papercite-data/pdf/issre13_dse.pdf

[27] BECK, Kent. Test-Driven Development By Example. Addison Wesley - Vaseem, 2003, ISBN 978-0321146533.

[28] GALTON, Francis. EUGENICS: ITS DEFINITION, SCOPE, AND AIMS. [online]. [cit. 12.4.2016]. Dostupný na WWW: https://web.archive.org/web/20071103082723/galton.org/essays/1900-1911/galton-1904-am-journ-soc-eugenics-scope-aims.htm

[29] CHEN, T.Y.; LEUNG, J.; MAK, I.K. Adaptive Random Testing [online]. [cit. 15.4.2016]. Dostupný na WWW: http://www.utdallas.edu/~ewong/SYSM-6310/03-Lecture/02-ART-paper-01.pdf

[30] GODEFROID, Patrice; KLARLUND, Nils; SEN, Koushik. Directed Automated Random Testing [online]. [cit. 15.4.2016]. Dostupný na WWW: https://wkr.io/public/ref/godefroid2005dart.pdf

[31] EVOSUITE. Commandline [online]. [cit. 20.4.2016]. Dostupný na WWW: http://www.evosuite.org/documentation/commandline/

[32] NEUVEDEN. GNU Lesser General Public License [online]. [cit. 20.4.2016]. Dostupný na WWW: https://cs.wikipedia.org/wiki/GNU_Lesser_General_Public_License

[33] FRASER, Gordon; ARCURI, Andrea. EvoSuite at the SBST 2013 Tool Competition [online]. [cit. 20.4.2016]. Dostupný na WWW: http://www.evosuite.org/wp-content/papercite-data/pdf/sbst13_competition.pdf

[34] PARASOFT, User's Manual for Jtest, version 9.6. Monrovia, CA 91016: Parasoft Corporation (2016).

[35] RANDOOP. Randoop manual [online]. [cit. 25.4.2016]. Dostupný na WWW: https://randoop.github.io/randoop/manual/index.html

[36] MYERS, Glenford J.; SANDLER, Corey; BADGETT, Tom. The Art of Software Testing, 3rd Edition. Hoboken, New Jersey: John Wiley & Sons, Inc., 2011, ISBN : 978-1-118-03196-4.