

Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra Počítačů

Framework pro testování platformy PCTgen

Lukáš Hopp

Leden 2017

Vedoucí práce: Ing. Miroslav Bureš Ph.D.

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Lukáš Hopp

Studijní program: Softwarové technologie a management
Obor: Softwarové inženýrství

Název tématu: Framework pro testování platformy PCTgen

Pokyny pro vypracování:

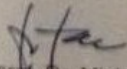
- 1) Navrhněte a implementujte Framework, pomocí kterého bude možné vytvářet automatizované uživatelské testy platformy PCTgen. Framework bude definovat architekturu testů tak, aby bylo možné je rychle vytvářet a efektivně udržovat.
- 2) Framework implementujte v Java s využitím vhodně vybraných existujících knihoven. Volbu těchto knihoven zdůvodněte.
- 3) Framework zdokumentujte. Součástí implementace bude sada min. 25 implementovaných ukázkových testů pro platformu PCTgen v tomto frameworku, pomocí kterých bude zároveň ověřena bezchybná funkčnost frameworku.

Seznam odborné literatury:

- [1] Bloch, J. Effective java (the java series). Prentice Hall PTR, 2008
- [2] Shildt, H. Java: the complete reference. McGraw-Hill, 2007

Vedoucí: Ing. Miroslav Bureš, Ph.D.

Platnost zadání do konce zimního semestru 2017/2018


prof. Dr. Michal Pěchouček, MSc.
vedoucí katedry




prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 12.9.2016

Poděkování / Prohlášení

Rád bych poděkoval mému vedoucímu bakalářské práce panu Ing. Miroslavu Burešovi Ph.D. za cenné rady, důležité připomínky a vstřícnost při vypracování bakalářské práce.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 10.1.2017

.....

Abstrakt / Abstract

Tato bakalářská práce se zabývá vývojem frameworku na vytváření automatizovaných uživatelských testů pro aplikaci PCTgen. Nejprve uvedu v práci problematiku testování softwaru a automatizovaných testů. Poté v práci provedu analýzu aplikace PCTgen. Následně vytvořím konkrétní návrh a implementaci frameworku, kde se zaměřím na testování Java swing aplikace s uživatelským rozhraním. Především se budu zabývat testováním grafů vytvořených za pomoci knihovny JGraphX. Tento vyvinutý framework bude snadno použitelný a efektivně udržovatelný. V poslední řadě budu tento framework demonstrovat na 25-ti testovacích scénářích naprogramovaných za pomoci zmíněného frameworku.

Klíčová slova:

Bakalářská práce, Automatizované uživatelské testy, Testování softwaru, Framework na vytváření testů, AssertJ, PCTgen

This bachelor thesis deals with developing framework for creating an automated user tests for application PCTgen. At first i will introduce software testing and automated tests. Then i will analyse PCTgen application. Then i will create specific design and implementation of framework, where I will focus on testing Java swing application with graphical user interface. I will mainly focus on testing graphs created by JGraphX library. This developed framework will be easy to use and efficiently maintainable. At least i will demonstrate this framework on 25 testcases programmed with help of mentioned framework.

Keywords:

Bachelor thesis, An automated user tests, Software testing, Framework for test creation, AssertJ, PCTgen

Obsah /

1 Úvod	1
2 Testování softwaru	2
2.1 Boehmův první zákon	2
2.2 Typy testů	3
2.2.1 Unitní testy	3
2.2.2 Regresní testy.	3
2.2.3 End to End testy.	3
2.2.4 Test driven develop- ment TDD.	3
2.2.5 Smoke testy	3
2.3 Automatizované testování	4
2.3.1 Uživatelské testy	4
2.3.2 Akceptační testy	4
2.3.3 Automatizované testování webových aplikací	4
2.3.4 Výhody a nevýhody automatických testů	4
2.3.5 Použití automatických a manuálních testů	5
3 Analýza	7
3.1 Aplikace PCTgen	7
3.1.1 grafické rozhraní	7
3.1.2 Technologie použité v aplikaci PCTgen	8
3.2 Požadavky	8
3.2.1 Funkční požadavky	8
3.2.2 Nefunkční požadavky	9
4 Návrh	10
4.1 Konceptuální návrh testova- cího frameworku	10
4.2 Výběr potřebných knihoven....	11
4.2.1 AssertJ	11
4.2.2 JUnit	11
4.3 Maven projekt.....	12
4.4 Framework jako samostatná aplikace.....	12
4.5 Klíčové třídy frameworku.....	13
5 Implementace	14
5.1 Programovací jazyk	14
5.2 interakce s grafem.....	14
5.3 Ukázka použití frameworku ...	14
5.4 Dokumentace nejdůležitěj- ších částí frameworku.....	18
5.4.1 Třída GuiEnd2EndTest..	18
5.4.2 Třída Project	19
5.4.3 Třída Graph	19
5.4.4 enum NodeCreation- Location.....	19
5.4.5 Třída Node.....	20
5.4.6 Třída TestUtils.....	20
5.4.7 TextConstants	21
5.5 Diagram nasazení.....	21
5.6 Spuštění testů.....	22
6 Vytvořené testovací scénáře	23
6.1 Vytvořit 30 nových projektů ..	23
6.2 Vytvořit 30 nových grafů v jednom projektu	23
6.3 Vytvořit 30 nových vrcholů v jednom grafu.	24
6.4 Vytvořit hrany střídavě v různých grafech	24
6.5 Vytvořit vrcholy střídavě v různých grafech	25
6.6 Vytvořit grafy střídavě v různ- ných projektech	25
6.7 Vygenerování testovacích scénářů z grafu bez hran a vrcholů	26
6.8 Vygenerování testovacích scénářů z grafu bez vrcholu start	26
6.9 Vygenerování testovacích scénářů z grafu bez vrcholu start	27
6.10 Každý vrchol v grafu musí mít příchozí hranu	27
6.11 Graf bez koncového vrcholu ...	28
6.12 Vytvoření grafu a ověření vygenerovaných testovacích scénářů	29
6.13 Validace grafu pomocí tla- čítka validate graph	30
6.14 Validace dialogu pro gene- rování testovacích scénářů	30
6.15 Očekávané oznámení o neu- ložení projektu	31
6.16 Neočekávané oznámení o ne- uložení projektu.	31
6.17 Defaultní crud model	32
6.18 Uložení projektu do souboru ..	32
6.19 Uložení projektu a následné nahrání projektu	32

6.20	Smoke test bez žádné priority u hran	33
6.21	Smoke test se stejnými prioritami na hranách.	33
6.22	Neaktuálních testovací scénáře	34
6.23	Validace validního grafu	35
6.24	Smazání vrcholu v grafu	35
6.25	Start v grafu může být jen jednou	36
7	Výsledky běhu vytvořených testů	37
8	Testování frameworku	39
9	Závěr	40
	Literatura	41
A	Obsah přiloženého cd	43
B	Požadavky na používání frameworku	44

Kapitola 1

Úvod

Testování aplikačního workflow je v dnešní době nejvíce používanou technikou, avšak manuální návrhy těchto testů jsou velmi náročné, jak časově, tak kvůli náchylnosti na lidský faktor v případě návrhu testovacího scénáře, z čehož následně vede náchylnost k chybám obzvláště při obsáhlých aplikačních workflow. Proto byl vyvinut nástroj na Fakultě Elektrotechnické na ČVUT. Tento nástroj[4] slouží k automatickému generování testovacích scénářů z nadefinovaného aplikačního workflow. PCTgen pracuje s algoritmy, které generují scénáře efektivně a optimálně. Lze si zvolit typ algoritmu, například smoketest algoritmus. Dále nástroj podporuje prioritizaci určitých cest v aplikačním workflow či hloubku pokrytí testu. Je zde také možné vytváření těchto workflow v zabudovaném editoru, buď jako orientovaný graf nebo jako activity diagram.

Cílem této práce je tedy navrhnout a implementovat framework¹⁾ na testování aplikace PCTgen a následně tento framework demonstrovat a otestovat na 25 testovacích scénářích. Jelikož testování aplikace je velmi důležité při vývoji a udržování softwaru, je vhodné mít sadu automatických regresních testů, které při každé změně či nové funkcionalitě ověří alespoň základní funkčnost aplikace z pohledu uživatele. Jelikož aplikace PCTgen je aplikace s grafickým rozhraním, tato práce bude řešit snadné vytváření end2end regresních testů, které budou simulovat uživatele. V práci se hlavně zaměřím na psaní testů, které budou moci vytvářet orientované grafy v zabudovaném editoru v aplikaci PCTgen. Vytvořená knihovna bude poskytovat api pro vývojáře, pomocí kterého bude možné snadno naprogramovat testy, které budou simulovat chování uživatele při používání aplikace PCTgen. Například pomocí myši a klávesnice automatizovaně klikat na definované komponenty, psát text do konkrétních textových polí či přesouvat různé objekty z bodu A do bodu B. To vše bude vytvořeno tak, aby bylo velmi jednoduché vyvíjení těchto testů.

V textu této bakalářské práce uvedu do problematiky testování softwaru. Popíši, kdy je vhodné automatizovat uživatelské testy. Zanalyzuji aplikaci PCTgen z hlediska použitých technologií, knihoven a také popíši grafické rozhraní aplikace. Následně zdokumentuji a popíši jednotlivé aspekty vývoje od návrhu až k implementaci. Zdokumentuji nejdůležitější části frameworku a ukážu použití frameworku na funkčním příkladě, který důkladně rozeberu a zdokumentuji. Také zde zdůvodním výběr knihoven třetích stran potřebných k vytvoření této práce. Nakonec popíši testovací scénáře, které tuto práci budou demonstrovat.

¹⁾ framework, lze také označit jako knihovnu, je soubor nástrojů, metod, datových struktur, které pomáhají řešit danou problematiku.

Kapitola 2

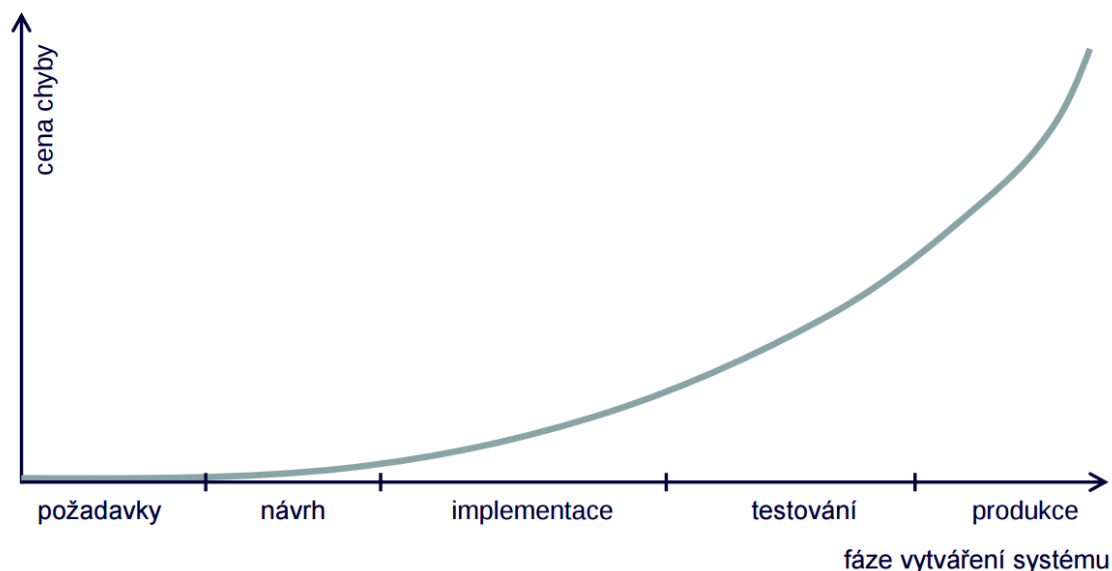
Testování softwaru

Testování softwaru je při samotném vývoji softwaru velmi podceňováno, ať už je důvodem například nedostatek času či nedostatečné finanční prostředky. Nikdo si však neuvědomuje, že pokud se při vývoji softwaru vyvíjejí i testy, může to naopak finanční prostředky a čas výrazně ušetřit. Testy jsou tedy velmi důležité a mohou nám pomoci v těchto aspektech:

- Udržitelnost softwaru
- Eliminace chyb
- Při udržování softwaru a vývoji nových funkcionalit pomáhá testování zabránit vzniku nových chyb pomocí regresních testů.
- Možnost vývoje technikou test driven development.
- Automatické testy mohou ušetřit zdroje manuálních testerů.
- Výrazné zlepšení kvality celého softwaru.

2.1 Boehmův první zákon

Chyby jsou nejčastější ve fázi požadavku a návrhu, a čím později odhalíme chybu v průběhu vývoje systému, tím je dražší [1].



Obrázek 2.1. Boehmův první zákon graficky zobrazen.¹⁾

Na grafu, který je na obrázku č. 2.1 je vidět, že cena chyby v každém vývojovém období značně roste. Proto je tedy vhodné testovat software v průběhu vývoje, abychom

¹⁾ Obrázek převzat z přednášek Ing. Miroslav Bureš Ph.D. pro předmět Testování softwaru.

zredukovali počet chyb v systému. Čím více chyb odhalíme v procesu vývoje, tím více času a prostředků ušetříme na opravování chyb v produkci.

2.2 Typy testů

Software lze testovat několika způsoby. Zde si popíšeme základní techniky, kterými se tato práce zabývá.

2.2.1 Unitní testy

Unitní testování je testování malých částí zdrojového kódu. Tyto testy píše programátor, aby ověřil funkčnost kódu. Testy například testují metodu v závislosti na různých vstupech a očekávají konkrétní výstupy, které ověřují. V jave je nejznámějším frameworkem na psaní unitních testů JUnit [2]. Tyto testy je následně vhodné spouštět při každém sestavení aplikace. Unitní testy však nejsou například vhodné při větším refaktorování kódu, kde se mění api a rozhraní. V takovém případě se musí tyto testy přepsat.

2.2.2 Regresní testy.

Pojem regresní test znamená, že při vývoji či udržování aplikace kontroluje určitá sada testů správnou funkčnost aplikace. V praxi to tedy znamená, že po vyvinutí nové funkčnosti ověří regresní testy, zda do softwaru nebyla zavlečena nějaká chyba. Dále například pokud se v aplikaci objeví chyba a programátor ji opraví, může na tuto chybu napsat regresní test, který i v budoucnu bude kontrolovat neopakující se vznik chyby. Jako regresní testy je nejvhodnější použít automatizované testy, které se nejlépe spouštějí při sestavení softwaru či při vložení práce do verzovacího repozitáře. Unit testy tedy lze používat jako regresní testy.

2.2.3 End to End testy.

End to End testy jsou testy, které ověřují funkčnost softwaru od spuštění přes určitou funkčnost až po ukončení aplikace. Simulují tedy uživatele, který software používá. Tento druh testování testuje software jako celek, a může proto sloužit i například jako integrační testy. Software může testovat na všech úrovních od grafického rozhraní přes aplikační logiku až po persistenci do databáze či souboru. V praxi se například tedy setkáme s end to end testy, které testují webové aplikace, kde testovací scénář zapsaný pomocí frameworku kliká myší, zapisuje vstupy do webové stránky a očekává změny či výstupy. V podstatě simuluje koncového uživatele, který má danou webovou aplikaci používat. Tento druh testování je velmi účinný, avšak velmi pracný a časově náročný.

2.2.4 Test driven development TDD.

Test driven development[3] je technika vývoje softwaru, kdy si vývojář nejdříve definuje, jak se má funkcionalita chovat a napíše test, který funkcionalitu ověřuje. Takový test spustí, čímž ověří, že funkcionalita není funkční. Poté programátor funkcionalitu naprogramuje a opět spustí test, který již má fungovat. Tento přístup má mnoho výhod a programátor tak píše čistší kód. Dále je tato technika výhodná při opravování defektů systému, kdy si vývojář nejdříve napíše test na chybu, která v systému je a dále ověří, že ji test odhalí. Následně tuto chybu opraví a ověří, zda test projde.

2.2.5 Smoke testy

Smoke testy jsou testy, které ověří základní funkčnost aplikace. Tyto testy se velmi často automatizují a také se velmi často spouštějí. Tyto testy jsou tak zásadní, že se v případě selhání tohoto testu, aplikace zpravidla dále netestuje.

2.3 Automatizované testování

Testování softwaru lze rozdělit na dvě kategorie. První kategorie závisí na tom, zda potřebují jednotlivé testy interakci s člověkem. Takovým testům říkáme manuální testy. Druhou kategorií jsou automatické testy, které vykonává samotný systém.

2.3.1 Uživatelské testy

Mezi manuální testy nejčastěji patří uživatelské testy. Tento typ testů spočívá v tom, že návrhář testů připraví testovací scénáře, které zpravidla obsahují konkrétní kroky popisující, jakým způsobem má tester interagovat se softwarem. Tento typ testů se používá u aplikací, které mají grafické rozhraní. Nejčastěji u webových aplikací nebo u tlustých klientů¹⁾. Tyto scénáře pak jsou nejčastěji aplikovány manuálně testery, kteří vykonávají krok po kroku podle testovacího scénáře.

2.3.2 Akceptační testy

Tento typ testů se používá k ověření funkčnosti softwaru při předávání softwaru od dodavatele k zadavateli. Zpravidla se aplikuje v poslední etapě vývoje. Na základě těchto testů zadavatel převezme software. Většinou je i tato skutečnost předmětem právních náležitostí mezi zadavatelem a dodavatelem. Dále se tento typ testů používá ve velkých společnostech, kde se akceptační testy využívají na ověření všech funkcností při vydání nových verzí. Tudíž lze tyto testy využívat i v rámci jedné společnosti, kdy zadavatelem i dodavatelem je stejná společnost. Tyto testy zpravidla navrhuje a provádí zadavatel. Akceptační testy se také spíše provádějí manuálně.

2.3.3 Automatizované testování webových aplikací

Nejčastěji automatizovanými testy jsou testy uživatelského rozhraní webových aplikací. To je způsobeno tím, že v současné době jsou právě webové aplikace nejvíce vyvíjeným softwarem. V závislosti na tuto poptávku vzniklo několik frameworků, které se zabývají právě automatizováním testů webových aplikací. Jeden z nejznámějších a nejpoužívanějších frameworků je Selenium web driver [11]. Tento framework slouží k automatizování uživatelských testů ve webovém prohlížeči a poskytuje api v několika programovacích jazycích. Dále tento framework podporuje nejpoužívanější webové prohlížeče.

2.3.4 Výhody a nevýhody automatických testů

Jelikož se v této práci zabývám především automatizováním uživatelských testů, budu se i v následujícím textu převážně zabývat automatizovanými testy uživatelského rozhraní. Pokud se tedy budeme rozhodovat, zda budeme některé uživatelské testy automatizovat, budou nás především zajímat výhody a nevýhody automatizovaných testů. Zde si některé výhody a nevýhody popíšeme. Mezi výhody automatizovaných testů patří zejména skutečnost, že cena provozu těchto testů je nejenom velice levná a zanedbatelná oproti manuálnímu testování, ale dokonce je i rychlost těchto testů výrazně vyšší. Tyto vlastnosti z nich dělají ideální nástroj k opakovanému spouštění.

Automatizované testy zároveň nedělají chyby a provádějí kroky testovacího scénáře vždy stejně. Manuální tester se oproti tomu může při vykonávání scénáře splést a některé důležité kroky zcela přeskočit, čímž následně vznikne pravděpodobnost neodhalení chyby. Mezi nevýhody automatických testů patří fakt, že automat nedokáže řešit neočekávané

¹⁾ Tlustý klient je aplikace, která má grafické rozhraní spouštěné přímo v systému. Na rozdíl od webové aplikace, kde grafické rozhraní je zobrazené na základě dat, které pošle server.

situace. Pokud se v systému objeví sebemenší změna, automatizovaný test v reakci na změnu pravděpodobně neprojde. Manuální tester má narozdíl od automatizovaných testů intuici a změna ho tudíž neovlivní natolik, aby testovací scénář nedokončil.

V úvodu této kapitoly jsem se zmiňoval o nepotřebnosti interakce s člověkem v průběhu automatizovaných testů. Toto platí v případě samotného běhu testu, avšak když je test zakončen negativním výsledkem, musí být nejprve tyto výsledky zkontrolovány člověkem (testerem), který může následně potvrdit chybu a případně jí zaslat vývojářům k opravě.

Samotný vývoj automatizovaných testů již testuje daný scénář a tudíž jedno z hlavních poslání automatizovaných testů je regrese, kdy se v budoucnu testuje a zamezuje opakovanému zavlečení chyb do systému.

Pokud máme testy automatizované, můžeme tyto testy spouštět na různých platformách a různých operačních systémech. Toto testování bychom v případě manuálních testů minimalizovali, jelikož by to bylo velmi nákladné. V automatických testech dále můžeme testovat velkou množinu vstupních dat, což by u manuálního testera trvalo velmi dlouho a bylo by to neefektivní.

Automatizované testy lze také použít na monitorování produkční aplikace, kdy v určitých intervalech automat spouští testy na produkční aplikaci, a tím monitoruje dostupnost[19].

■ 2.3.5 Použití automatických a manuálních testů

Otázkou tedy je, za jakých podmínek použít manuální testy či automatizované testy? Následující kritéria by tuto otázku měla zodpovědět.

■ 2.3.5.1 Frekvence spouštění testů

Pokud budeme mít regresní testy, které nám budou ověřovat při každém sestavení softwaru funkčnost a tyto testy tedy budeme spouštět velmi často, je velmi výhodné tyto testy zautomatizovat. Naopak, pokud budeme mít testovací scénář, který otestuje určitou novou funkčnost a nebo opravení chyby jen jednou či dvakrát za životnost softwaru, je tento typ testu vhodnější testovat manuálně. Jelikož by bylo nákladnější tento test vyvinout a v budoucnu udržovat.

■ 2.3.5.2 Typ testů

Dále závisí na typech testů. Například unitní testy jsou z pohledu vývoje velmi levné a účinné, tudíž jsou tyto testy nejčastějšími automatickými testy. Na druhou stranu by bylo velmi nákladné mít každou funkcionalitu v aplikaci otestovanou automatizovaným end to end regresním uživatelským testem. Proto musíme najít kompromis a otestovat jen hlavní funkcionality a nejdůležitější části aplikace. Vhodné je tedy vytvořit automatizované uživatelské smoke testy, kdy se z pohledu uživatele ověří základní funkčnost aplikace.

■ 2.3.5.3 Podle testované části softwaru

Zde se musíme zamyslet nad tím, zda se určitým částem aplikace bude v budoucnu často měnit uživatelské rozhraní. Pokud se uživatelské rozhraní bude často měnit, je lepší tyto části testovat manuálně, jelikož udržování automatických testů by bylo nákladné.

■ 2.3.5.4 Typ testovaného softwaru

Dále je třeba si zanalyzovat, zda existují na trhu prostředky, které nám vývoj automatizovaných uživatelských testů usnadní a zda se nám vyplatí tyto testy vyvíjet. Pokud bychom chtěli vyvinout uživatelské testy na webové aplikace, je zde mnoho možností

a navíc není vývoj těchto testů nijak zvlášť nákladný. Naopak pro testování grafického rozhraní tlustých klientů zde moc prostředků není a současně také závisí na platformě. Pokud tedy zjistíme, že neexistuje specializovaný framework na testování určité platformy, musíme volit univerzální frameworky na vytváření klikacích robotů. Toto řešení je však časově náročné a nestabilní.

Kapitola 3

Analýza

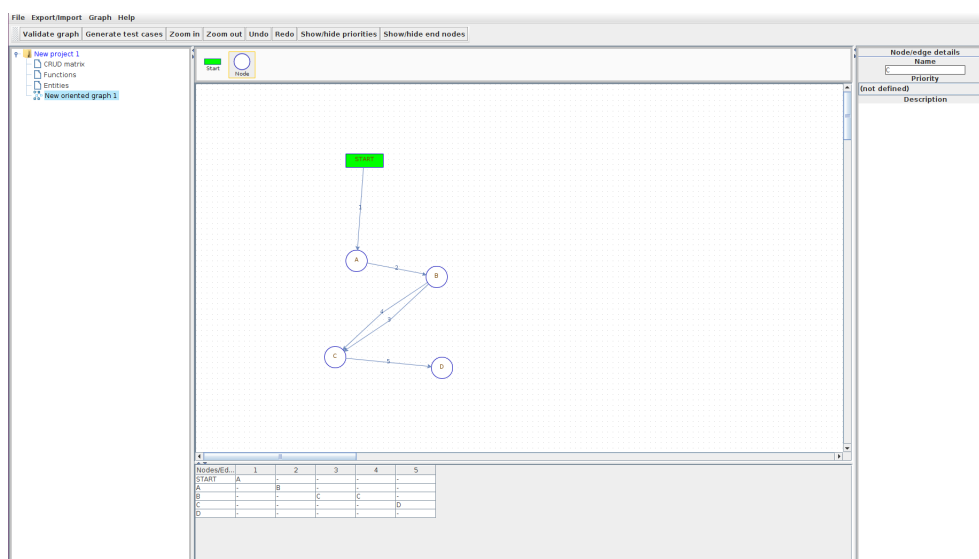
V této kapitole především zanalyzuji aplikaci PCTgen. Zanalyzuji veškeré technologie, které jsou použité a následně se zaměřím na grafické rozhraní, které je nejdůležitějším aspektem pro tuto práci.

3.1 Aplikace PCTgen

O aplikaci PCTgen jsem se již zmiňoval v úvodu této práce. Aplikace PCTgen tedy slouží k nadefinování a modelování aplikačního workflow s následným vygenerováním testovacích scénářů. Aplikace navíc obsahuje několik zajímavých funkcionalit, mezi které například patří podpora importu a exportu aplikačního workflow, a je tedy možné importování workflow vytvořeného v jiném nástroji a případné vygenerování testovacích scénářů.

3.1.1 grafické rozhraní

Grafické rozhraní (gui¹⁾) aplikace PCTgen lze rozdělit na několik částí. V levé části se nachází stromová struktura projektu, kde jsou zobrazeny komponenty, které patří do příslušného projektu. Uprostřed se nachází komponenta, která slouží k modelování orientovaných grafů či activity diagramů. v dolní části gui se nachází tabulka, která popisuje výčet hran a vrcholů orientovaného grafu. V pravé části je komponenta, ve které lze nastavovat různé vlastnosti objektů, se kterými právě pracujeme. Například pokud pracujeme s orientovaným grafem, lze u hran nastavit priorita či popis hrany. Dále gui obsahuje paletu akcí, které lze na objekty či model použít. Tyto akce se mění v závislosti na objektu, s kterým právě uživatel pracuje.



Obrázek 3.1. Grafické rozhraní aplikace PCTgen

¹⁾ zkratka pro graphical user interface neboli grafické rozhraní

■ 3.1.2 Technologie použité v aplikaci PCTgen

Aplikace prctgen je naprogramovaná v Javě 8 a využívá následující technologie a knihovny.

■ 3.1.2.1 Swing

Swing[5] je knihovna pro tvorbu grafického rozhraní a je součástí javy. Za pomoci této knihovny lze vytvářet grafické rozhraní, kterým uživatel ovládá aplikaci. Mezi komponenty, které lze vytvářet patří například okna, dialogy, tlačítka, lišty a další. Grafické rozhraní aplikace PCTgen je naprogramovaná za pomoci této knihovny.

■ 3.1.2.2 Maven

Maven[6] je nástroj na management softwarového projektu. Pomocí mavenu lze definovat, jak se bude projekt sestavovat. Dále lze pomocí tohoto nástroje jednoduše udržovat a definovat závislosti na modulech či knihovnách. Všechny tyto definice se definují v souboru zvaném pom¹). To je výhodné při nasazování nových verzí použitých knihoven, kdy se v souboru pom jen změní verze u jednotlivých knihoven a maven tyto knihovny při následném sestavení stáhne ze vzdáleného serveru. Také lze definovat různé úkoly a tyto úkoly spouštět při sestavení. Mezi tyto úkoly například patří různé kopírování souborů, spuštění testů, sestavení samostatných částí systému a další.

■ 3.1.2.3 JGraphX knihovna

JGraphX[7] je opensource knihovna napsaná v jave na zobrazování a práci s grafy či různými diagramy. Především se zaměřuje na interakci s diagramy složenými z vrcholů a hran. Za pomoci této knihovny lze programovat aplikace na modelování a zobrazování aplikačního workflow, uml diagramů, modelování elektronických obvodů či vizualizaci počítašové sítě. Dále knihovna obsahuje několik funkcionalit na podporu importu či exportu těchto modelů a diagramů. Za pomoci této knihovny, jsou v aplikaci PCTgen naimplementovány grafy a activity diagramy.

■ 3.2 Požadavky

V této kapitole uvedu požadavky na testovací framework dle standardní metodiky, která požadavky rozděluje na funkční a nefunkční.

■ 3.2.1 Funkční požadavky

V této podkapitole udělám výčet funkčních požadavků na framework a také mezi tyto požadavky zahrnu oblasti aplikace, které bude testovat zadaných 25 testovacích scénářů.

- Vytvoření frameworku na testování aplikace PCTgen.
- Vytvoření frameworku na vytváření uživatelských regresních testů.
- Podpora editoru na vytváření grafů.
- Podpora základních operací v aplikaci PCTgen.
- Vytvoření sady regresních testů.
- Vytvoření testů na základní funkčnost aplikace PCTgen.
- Vytvoření testů na automatické pojmenování objektů jako například grafy.
- Vytvoření smoke testu.
- Vytvoření testů na validaci grafu.
- Vytvoření testů na validní a nevalidní vstupy do textových polí.

¹) project object model

- Vytvoření testů na vyskakovací dialogové okna.
- Vytvoření testů na různé algoritmy generující testovací scénáře.

■ 3.2.2 Nefunkční požadavky

V této podkapitole udělám výčet nefunkčních požadavků.

- Nezávislý na zdrojovém kódu PCTgen.
- Framework naprogramovaný v java 8.
- Vhodné použití knihoven třetích stran.
- Opakovatelný běh testů.
- Snadno rozšiřitelné.
- Efektivně udržovatelné.
- Snadno zakomponovatelné do zdrojového kódu PCTgen.

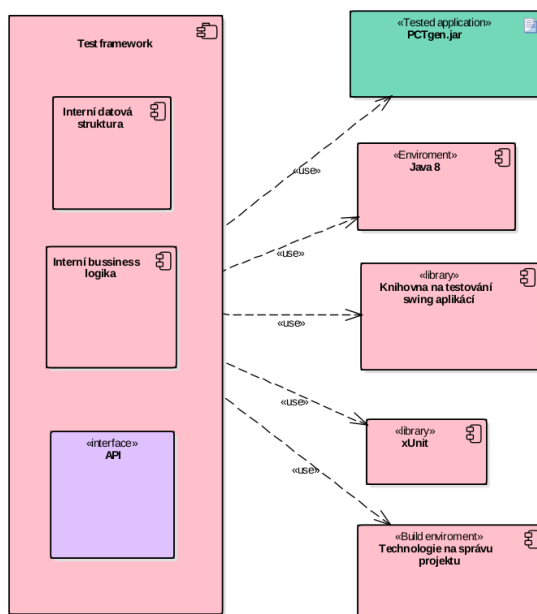
Kapitola 4

Návrh

Testovací framework, jsem navrhnul tak, aby bylo možné vytvářet automatizované end to end regresní testy. Programátor napíše v programovacím jazyce Java, za pomoci tohoto frameworku, testovací scénář, který simuluje uživatele od zapnutí aplikace, přes samotný testovací scénář až po ukončení aplikace. Vše bude automatizované a při vývoji tedy lze tyto testy spouštět například při každém commitu do verzovacího repozitáře. Tím se programátor může ujistit, že nikde nezalekl chybu. Mým cílem bylo také vytvoření frameworku, který by pomohl při vývoji či udržování aplikace PCTgen s aplikováním test driven developmentem. Tzv programátor naprogramuje testovací scénář a spustí jej, aby se ujistil, že test neprojde a následně naimplementuje novou funkcionalitu nebo opraví chybu a nakonec si ověří, zda test prošel. Při návrhu jsem se hlavně řídil pravidlem, že veškerý kód by měl být programován tak, jakoby se jednalo o knihovnu, kterou by používali jiní programátoři. Toto platí i v případě že se o knihovnu nejedná[14].

4.1 Konceptuální návrh testovacího frameworku

Při návrhu testovacího frameworku jsem si nejdříve nadefinoval z jakých částí se framework bude skládat. Tento návrh můžeme vidět na obrázku č. 4.1. Zde popíši, proč jsem vybral tyto konkrétní části.



Obrázek 4.1. Konceptuální návrh testovacího frameworku.

Programovací jazyk, ve kterém tento framework vyvinu, byl již zadán jako nefunkční požadavek. Dále jsem zde uvedl nástroj na správu projektu a závislostí. Ten bude určen k snadnému udržování externích knihoven, což je již dnes standardem. Jedna z hlavních částí frameworku bude vhodná knihovna, která nám usnadní testování swingových komponent [5]. Pro snadné porovnávání různých objektů použiji knihovnu typu xUnit, která slouží na psaní unitních testů. Samozřejmě zde musím uvést i samotnou aplikaci PCTgen, která se bude do frameworku vkládat jako jar¹⁾ soubor. V poslední řadě zde bude samotný framework, který se bude skládat z interní datové struktury, interní logiky a api/fnotezkratka pro Application Programming Interface, jde o sbírku metod, funkcí, datových struktur, které poskytuje., kterou bude poskytovat vývojářům.

4.2 Výběr potřebných knihoven.

4.2.1 AssertJ

Na trhu je velmi málo testovacích frameworků, který dokážou testovat swingové aplikace [5]. Tudíž výběr této knihovny, která je nejdůležitější externí knihovnou v této práci, nebyl složitý. Existuje totiž velmi kvalitní a propracovaný soubor různých testovacích frameworků, které usnadňují testování různých částí typů Java aplikací. Tento soubor testovacích scénářů se jmenuje AssertJ[9] a obsahuje právě konkrétní knihovnu na testování grafického rozhraní naprogramovaného pomocí knihovny swing v Javě [5]. Tato konkrétní knihovna se jmenuje assertJ-swing[9] a její hlavní výhody jsou :

- *Simulace interakce uživatele s grafickým rozhráním, například pomocí drag 'n drop.*
- *Vyhledávání grafických komponent pomocí typu, jména či vlastního kritéria.*
- *Podpora všech swingových komponent, které jsou v standardní knihovně v jdk.*
- *Kompaktní a výkonné API na vytváření a udržování funkcionálních GUI testů.*
- *Podpora applet testů.*
- *Podpora fotek obrazovky u neúspěšných testů v HTML test reportech.*
- *Lze použít s testovacími frameworky JUnit a TestNG.*
- *Podpora testování porušování swingových pravidel jak použít vlákna. [9]*

Další velkou výhodou této knihovny je, že je v současné době udržována a stále vyvíjena.

Jako alternativní knihovny, které by bylo možné použít, jsou knihovny používané na uživatelské testy psané v jazyce Javascript, například knihovna sikuli[8]. Dokonce tato knihovna má vytvořené api pro vytváření těchto testů v jazyce Java. Jelikož, ale existuje knihovna, která je přímo na testování swingových aplikací [5], bylo by použití této knihovny náročnější.

4.2.2 JUnit

Knihovna JUnit je nejrozšířenější framework na programování unitních testů v programovacím jazyce Java. Proto jsem tedy vybral tuto knihovnu do této práce, abych mohl použít její API na porovnávání očekávaných výstupu a reálných výstupu v testovacích scénářích. Mezi další nepoužívanější frameworky patří testNG[12] avšak, pro účely tohoto frameworku je vhodnější použít JUnit, jelikož je také použit v aplikaci PCTgen a v budoucnu by bylo snazší pro vývojáře JUnit používat i v tomto frameworku.

¹⁾ *.jar je souborový formát, používaný javou.

4.3 Maven projekt

Na trhu jsou dvě nejpoužívanější technologie na správu softwarových projektů - gradle[13] a maven[6]. Obě tyto technologie jsou dostatečné na správu této práce, avšak jelikož samostatná aplikace PCTgen je také spravována pomocí mavenu, rozhodl jsem se maven použít i v této práci na správu projektu a na správu závislostí externích knihoven. Výhodné to bude v budoucnu pro vývojáře PCTgenu, kteří by chtěli využívat tento testovací framework. Ve frameworku bude sloužit k snadnému definování externích knihoven, které se následně při vývoji automaticky stáhnou ze vzdálených repositářů. Dále s touto technologií je jednodušší spravovat nové verze těchto knihoven, oproti klasickému přístupu, kdy se musí tyto knihovny vložit přímo do projektu.

4.4 Framework jako samostatná aplikace

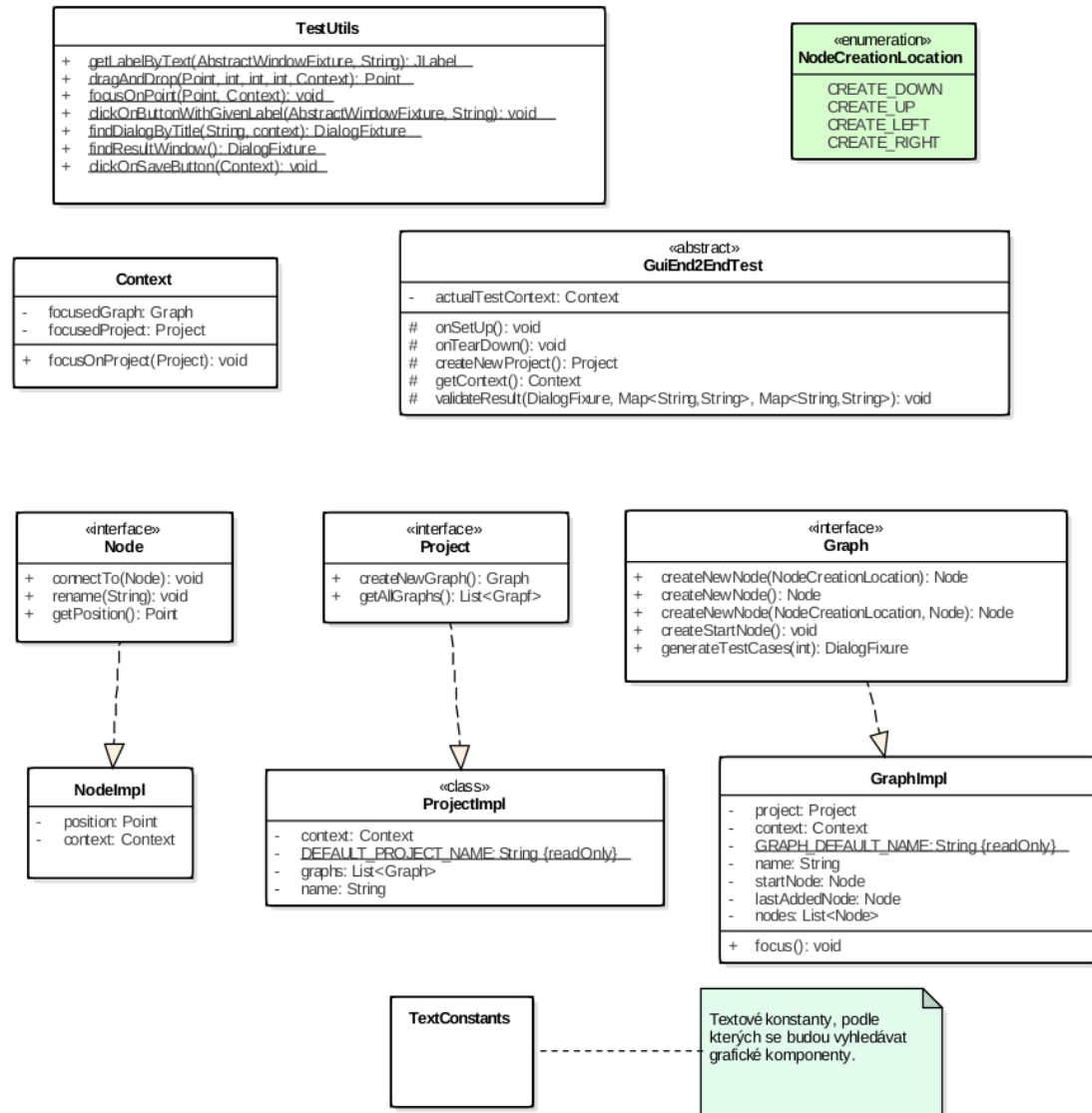
Jelikož se v průběhu vývoje tohoto testovacího frameworku paralelně vyvíjela i aplikace PCTgen, musel jsem z následujících důvodů tento framework vyvinout jako samostatnou aplikaci, do které se vloží spustitelný jar soubor aplikace PCTgen.

- Architektura aplikace se velmi rozsáhle předělávala.
- Vývoj aplikace je verzován v gitu a vývojáři, kteří vyvíjí tuto aplikaci, by museli neustále řešit konflikty s mým vývojem frameworku.

Pokud by se tento framework v budoucnu měl využívat jako součást vývoje aplikace PCTgen, bylo by zcela jednoduché tento testovací framework vložit jako modul do projektu PCTgen.

4.5 Klíčové třídy frameworku

V této kapitole popíší diagram nejdůležitějších tříd v testovacím frameworku. Samotný testovací framework lze rozdělit na tři části, o kterých jsem se již zmínil v kapitole o konceptuálním návrhu na obrázku č. 4.1. Dále tedy v této kapitole popíší, které třídy patří do jednotlivých částí.



Obrázek 4.2. Klíčové třídy frameworku.

Začnu interní datovou strukturou, která se skládá z rozhraní Graph, Project, Node a implementací GraphImpl, ProjectImpl a NodeImpl. Tato datová struktura je nezbytná pro práci s grafy a pro pohodlné psaní uživatelských testů. Vše podrobněji ukáží v následujících kapitolách. Další částí je samostatná logika, do které patří TextUtils, Context a TextConstants. Tyto třídy jsou interní záležitostí, usnadňují práci s externími knihovnami a adaptují je na aplikaci PCTgen. Poslední částí je samostatné API pro uživatele tohoto frameworku. Zde je nejdůležitější třídou GuiEnd2EndTest. Tato třída bude abstraktním testovacím scénářem, který vše připraví a poskytne metody, které budou sloužit k vytváření uživatelských testů.

Kapitola 5

Implementace

V kapitole popíši důležité aspekty, na které jsem narazil při implementaci frameworku. Popíši a okomentuji zde ukázkou použití frameworku, která bude demonstrovat vytváření grafů v aplikaci pctgen. Zdokumentuji zde nejdůležitější části frameworku a nakonec vysvětlím, jakým způsobem lze framework nasadit a automatické testy spustit.

5.1 Programovací jazyk

Aplikace PCTgen je naprogramovaná v programovacím jazyce Java ve verzi 1.8. Z tohoto důvodu je nejsnazší a zároveň nejefektivnější i tento framework naprogramovat v programovacím jazyce Java. Další výhodou je, že vývojář, který bude vyvíjet a udržovat aplikaci, bude znát programovací jazyk Java, a proto pro něho bude i snazší používat tento framework.

5.2 interakce s grafem

Nejtěžším úkolem testovacího frameworku bylo vymyslet, jakým způsobem bude implementována práce s grafem. Jelikož knihovna `assertJ-swing` [9] podporuje jen standardní komponenty, které obsahuje JDK¹⁾ a nemá tedy podporu knihovny `JGraphX`, která je použita pro práci s grafy v aplikaci PCTgen. Musel jsem tedy naimplementovat veškerou práci s grafovými komponenty sám. To jsem naimplementoval tak, že si framework zmapuje komponenty, které lze do grafu přesunout a také si zmapuje, jak je velké plátno, do kterého lze tyto komponenty přetáhnout. Přesouvání těchto komponent je naimplementováno tak, že kurzor myši se přesune na pozici objektu, který chceme přesunout do plátna a následně je spočítána trasa na úrovni bodů na obrazovce. Po této trase je následně s určitou rychlostí tento objekt přesunut a konkrétní pozice objektu uložena. Pozice těchto objektů jsou dále využívány například pro spojování vrcholů hrany, či přesouvání vrcholů. Od této implementace je však uživatel tohoto frameworku odstíněn takovým způsobem, že si jen zadefinuje, jakou má mít nový vrchol v grafu pozici vůči jinému vrcholu.

5.3 Ukázka použití frameworku

V této kapitole ukážu použití frameworku na funkčním příkladu zdrojového kódu. Kompletní kód si rozdělím na jednotlivé části, které následně popíši.

Potřebné knihovny a balíčky pro tuto ukázkou.

¹⁾ Java development kit je potřebný nástroj na vývoj aplikací v javě.

```

package end2endtests;

import java.util.HashMap;
import java.util.Map;

import end2endtestframework.Graph;
import end2endtestframework.GuiEnd2EndTest;
import end2endtestframework.Node;
import end2endtestframework.Project;
import org.assertj.swing.fixture.DialogFixture;
import org.junit.Test;

```

Použití je velmi jednoduché. Pro vytvoření testovacích scénářů stačí vytvořit třídu, která dědí od třídy `GuiEnd2EndTest`. Třída `GuiEnd2EndTest` se již postará o spuštění aplikace `PCTgen` a poskytuje API pro vytváření testovacích scénářů. Je velmi vhodné vytvořit jednu třídu pro více podobných testovacích scénářů. Metody, které jsou oannotovány anotací `@Test`, jsou jednotlivé testovací scénáře. Při každém spuštění těchto metod třída `GuiEnd2EndTest` zajistí znovuspuštění aplikace `PCTgen`.

```

/**
 * @author Lukas Hopp
 */
public class TestFrameworkExample extends GuiEnd2EndTest {

```

Testovací scénář s názvem `example1`.

```

@Test
public void example1() throws InterruptedException {

```

Tato část demonstruje vytváření nového projektu a v tomto projektu vytvoření nového grafu.

```

// Vytvoří projekt.
Project project1 = createNewProject();

// Vytvoří graf v projektu.
Graph graph1 = project1.createNewGraph();

```

Zde je demonstrováno, jak je jednoduché vytvářet vrcholy v grafu. Tyto vrcholy lze vytvářet pomocí metody `createNewNode()`, kdy se tyto vrcholy automaticky vloží pod poslední vložený vrchol. Druhá možnost je vkládat vrcholy na konkrétní pozici určenou lokací oproti konkrétnímu vrcholu. Tyto vrcholy lze vytvářet na různých grafech v různých projektech. V tomto příkladu vkládáme vrcholy do grafu s názvem `graph1`.

```

// Vytvoří vrchol start
Node start = graph1.createStartNode();

```

```

// Ostatní vrholy
Node a = graph1.createNewNode();
Node b = graph1.createNewNode();
Node c = graph1.createNewNode();

// Vytvoří vrchol v grafu nalevo od vrcholu c.
Node d = graph1.createNewNode(
Graph.NodeCreationLocation.CREATE_LEFT, c);

// Vytvoří vrchol v grafu napravo od vrcholu c.
Node e = graph1.createNewNode(
Graph.NodeCreationLocation.CREATE_RIGHT, c);

```

Vytvořené vrholy lze orientovaně spojit hranou. V ukázce například pomocí příkazu *a.connectTo(b)* spojíme vrchol *a* s vrcholem *b*, kdy hrana bude směrem od *a* do *b*. Směr této hrany lze vytvořit i opačně, pokud bychom příkaz použili takto *b.connectTo(a)*.

```

// Spojení vrcholu orientovanými hranami.
start.connectTo(a);
a.connectTo(b);
b.connectTo(c);
c.connectTo(d);
c.connectTo(e);
d.connectTo(a);

```

Pokud již máme vytvořený graf a chceme ověřit, zda se pro tento graf správně vygenerují testovací scénáře. Zavoláme metodu *generateTestCases(1)* na příslušném grafu a jako vstupní argument vložíme požadovanou hloubku testovacího scénáře. Metoda klikne na tlačítko v aplikaci, které vygeneruje testovací scénáře a dialog s testovacími scénáři nalezne a uloží do proměnné s názvem *dialogFixture*.

```

// vygeneruje testovací scénáře
final DialogFixture dialogFixture = graph1.generateTestCases(1);

```

Na konci tohoto testu si připravíme očekávané výsledky vygenerovaných testovacích scénářů na základě vytvořeného grafu. Aplikace PCTgen zobrazí dialogové okno se dvěma záložkami. V první záložce je tabulka, ve které jsou kombinace hran a vrcholů. V druhé záložce jsou vygenerované testovací scénáře. V tabulkách jsou výsledky popsány stejným způsobem jako mapy očekávaných výsledků v této ukázce. U kombinací je použit automaticky vygenerovaný název vrcholů velkými písmeny abecedy a hran číslovaných 1 až N. Klíčem mapy je tedy název hrany a hodnotou jsou hrany, které jsou spojené s tímto vrcholem. Na posledním řádku zavoláme metodu *validateResult*, která zvaliduje očekávané výsledky s výsledky v aplikaci.

```

// A,B,C,D .. je standardní pojmenování
// nových vrcholů v aplikaci PCTgen
// Zde je vytvořen očekávaný výsledek vrcholu a hran.
final Map<String, String> expectedEdgesResult =
new HashMap<String, String>() {{
    put("A", "1,6");
    put("B", "2");

```



```

        put("C", "3");
        put("D", "4");
        put("E", "5");
    });

    // Zde je očekávaná sekvence hran testovacího scénáře
    final Map<String, String> expectedTestSituations =
    new HashMap<String, String>() {{
        put("1", "1 - 2 - 3 - 4 - 6 - 2 - 3 - 5");
    }};

    // Zvaliduje očekávaný výstup aplikace.
    validateResult(dialogFixture, expectedEdgesResult,
        expectedTestSituations);
}

```

Následující ukázka demonstruje, jakým způsobem je uživatel frameworku odstíněn od akcí v aplikaci PCTgen při práci s grafy a projekty. Za pomoci frameworku, lze v aplikaci pracovat s několika projekty a grafy v jednom testovacím scénáři. V následujícím bloku kódu se vytvoří první projekt a v tomto projektu se vytvoří orientovaný graf.

```

@Test
public void example2() throws InterruptedException {

    //----- První projekt -----
    // Vytvoří první projekt.
    Project project1 = createNewProject();

    // Vytvoří první graf v prvním projektu.
    Graph graph1 = project1.createNewGraph();

    // Vytvoří vrcholy v grafu.
    Node start = graph1.createStartNode();
    Node a = graph1.createNewNode();
    start.connectTo(a);
}

```

Zde vytvoříme druhý projekt a v projektu vytvoříme druhý graf. Při každé akci na určitém projektu či grafu si framework zjistí, zda se s tímto objektem aktuálně pracuje a nebo je nutné se do tohoto grafu či projektu přepnout. Tento přístup tedy uživatele odstíní od mnoha akcí a je jednodušší a rychlejší psát testy.

```

//----- Druhý projekt -----
// Vytvoření druhého projektu.
Project project2 = createNewProject();

// Zde se automaticky přepne na druhý graf v levé části aplikace.
Graph graph2 = project2.createNewGraph();

// V novém grafu se vytvoří nový vrchol.
Node start2 = graph2.createStartNode();

// Přidání vrcholu do prvního grafu
}

```

```

//Automaticky se přepne na první graf.
Node b = graph1.createNewNode();
a.connectTo(b);

// Vytvoření druhého grafu v druhém projektu.
// Automaticky se přepne na druhý projekt
// a zde vytvoří nový graf.
Graph graph2_2 = project2.createNewGraph();
graph2_2.createStartNode();

// Vygenerují se testovací scénáře.
graph1.generateTestCases(1).close();
}
}

```

5.4 Dokumentace nejdůležitějších částí frameworku

5.4.1 Třída *GuiEnd2EndTest*

Tato třída slouží k vytvoření testovacího scénáře. Uživatel vytvoří svoji třídu, která bude reprezentovat testovací scénáře a bude dědit od této třídy *GuiEnd2EndTest*. Tato třída při každém běhu metody, která je anotovaná anotací *@Test*, spustí aplikaci *PCTgen*, která je dále testována testovacím scénářem naprogramovaným uvnitř metody. Je určitě vhodné mít třídu pro více podobných testovacích scénářů. Například třídu, kde budou testovací scénáře, které budou testovat textové pole v jednom dialogovém okně.

metody :

```
protected Project createNewProject()
```

Metoda vytvoří nový projekt a vrátí objekt *Projekt*, na kterém následně lze provádět další operace. V grafickém rozhraní aplikace to tedy znamená, že framework klikne na tlačítko *file* a dále vybere položku *new project*.

```
protected void validateResult(
    DialogFixture dialogWithGeneratedTestCases,
    Map<String, String> expectedEdges,
    Map<String, String> expectedTestCases
)
```

Metoda porovná vygenerovaný výsledek testovacích situací vygenerované v aplikaci s očekávaným výsledkem definovaným v testovacím scénáři. Parametr *dialogWithGeneratedTestCases* je dialog zobrazený aplikací, ve kterém jsou výsledky vygenerovaných testovacích situací. Tento dialog lze například získat z objektu *Graph* takto : *graph.generateTestCases(1)*. Parametr *expectedEdges* je mapa očekávaných kombinací hran a vrcholů. Parametr *expectedTestCases* je mapa očekávaných posloupností hran v testovacím scénáři.

Příklad hodnoty v mapě očekávaných posloupností hran v testovacím scénáři číslo jedna. Key: 1 (Testovací situace číslo jedna), Value: 1 - 2 - 3 - 4 - 6 - 2 - 3 - 5 (posloupnost hran v testovacím scénáři).

■ 5.4.2 Třída Project

Tato třída reprezentuje projekt v aplikaci, který může obsahovat grafy a jiné objekty. Na tomto objektu dále lze vytvářet v projektu nové grafy.

metody :

```
public Graph createNewGraph()
```

Metoda vytvoří nový graf v projektu, na kterém je tato metoda volána. V aplikaci klikne pravým tlačítkem myši na projekt a vybere položku new graph. Dále metoda vrátí nově vytvořený graf v projektu.

```
public String getName()
```

Metoda vrátí jméno projektu.

■ 5.4.3 Třída Graph

Tato třída reprezentuje graf v projektu. Tento graf již může obsahovat vrcholy a hrany. Z tohoto grafu lze vygenerovat testovací situace.

metody :

```
public Node createStartNode()
```

Tato metoda vytvoří vrchol start. V aplikaci tedy přesune vrchol start z horní lišty do plátna uprostřed. Dále metoda vrátí nově vytvořený vrchol start.

```
public Node createNewNode()
```

Metoda vytvoří nový vrchol typu node pod posledním vloženým vrcholem v grafu. Dále metoda vrátí nově vytvořený vrchol.

```
public Node createNewNode(NodeCreationLocation location)
```

Vytvoří nový vrchol typu node definovaný lokací vzhledem k poslednímu vloženému vrcholu. Parametr location je lokace vzhledem k poslednímu vloženému vrcholu. Metoda vrátí nově vytvořený vrchol.

```
public Node createNewNode(NodeCreationLocation location,
    Node positionComparedTo)
```

Vytvoří nový vrchol definovaný lokací vzhledem k definovanému vrcholu. Parametr location je lokace vzhledem k definovanému vrcholu parametrem positionComparedTo. Metoda vrátí nově vytvořený vrchol

■ 5.4.4 enum NodeCreationLocation

Tento enum slouží k snadnému vkládání nových vrcholů do grafu vzhledem k ostatním vrcholům. Jsou možné tyto hodnoty :

```
CREATE_DOWN
```

Za pomoci této hodnoty lze vkládat vrchol do grafu pod požadovaný vrchol. Vrchol bude na vertikální ose pod požadovaným vrcholem.

CREATE_LEFT

Za pomoci této hodnoty lze vkládat vrchol do grafu vlevo od požadovaného vrcholu. Vrchol bude na horizontální ose vlevo od požadovaného vrcholu.

CREATE_RIGHT

Za pomoci této hodnoty lze vkládat vrchol do grafu vpravo od požadovaného vrcholu. Vrchol bude na horizontální ose vpravo od požadovaného vrcholu.

CREATE_UP;

Za pomoci této hodnoty lze vkládat vrchol do grafu nad požadovaný vrchol. Vrchol bude na vertikální ose nad požadovaným vrcholem.

■ 5.4.5 Třída Node

Tato třída reprezentuje vrchol v grafu. Každý takový vrchol má určitou pozici v grafu a lze jej spojovat s ostatními vrcholy.

metody :

```
public void connectTo(Node node)
```

Metoda spojí hranou vrchol, na kterém je tato metoda zavolána s vrcholem, který je vstupním argumentem node.

■ 5.4.6 Třída TestUtils

V této třídě jsou pomocné metody, které usnadňují práci s knihovnou assertJ [9] a adaptují tuto knihovnu na aplikaci PCTgen.

metody :

```
public static JLabel getLabelByText(AbstractWindowFixture mainFrame,
                                   final String text)
```

Metoda najde a vrátí objekt JLabel podle textu v něm. Vstupním parametrem mainFrame musí být hlavní okno aplikace. Parametrem text je text, podle kterého se má JLabel nalézt. Metoda vrátí objekt JLabel.

```
public static Point dragAndDrop(Point start, int pxMoveDown,
                                int pxMoveXAxis, int delayBeforeDrag,
                                GuiEnd2EndTest.Context context)
```

Tato metoda přesune objekt, který je na vstupním parametru start o počet pixelů na vertikální ose a o počet pixelů na horizontální ose. Parametr start je poloha počátečního bodu. Parametr pxMoveDown je počet pixelů na horizontální ose o kolik se má objekt posunout. Parametr pxMoveXAxis je počet pixelů na vertikální ose o kolik se má objekt posunout. Parametr delayBeforeDrag je zpoždění před zahájením posunu. Parametr context je kontext testovacího scénáře. Metoda vrátí polohu koncového bodu.

```
public static void clickOnButtonWithGivenLabel(
                                   AbstractWindowFixture window,
                                   final String label)
```

Metoda klikne levým tlačítkem myši na tlačítko s textem. Parametr window je okno, kde se má nacházet tlačítko. Parametr label je text, který má být na tlačítku.

```
public static DialogFixture findDialogByTitle(
                                   final String title,
                                   GuiEnd2EndTest.Context context)
```

Najde dialogové okno podle titulku. Parametr title je text titulku okna. Parametr context je kontext testovacího scénáře. Metoda vrátí dialogové okno, které našla podle titulku.

```
public static DialogFixture findResultWindow(
    GuiEnd2EndTest.Context context)
```

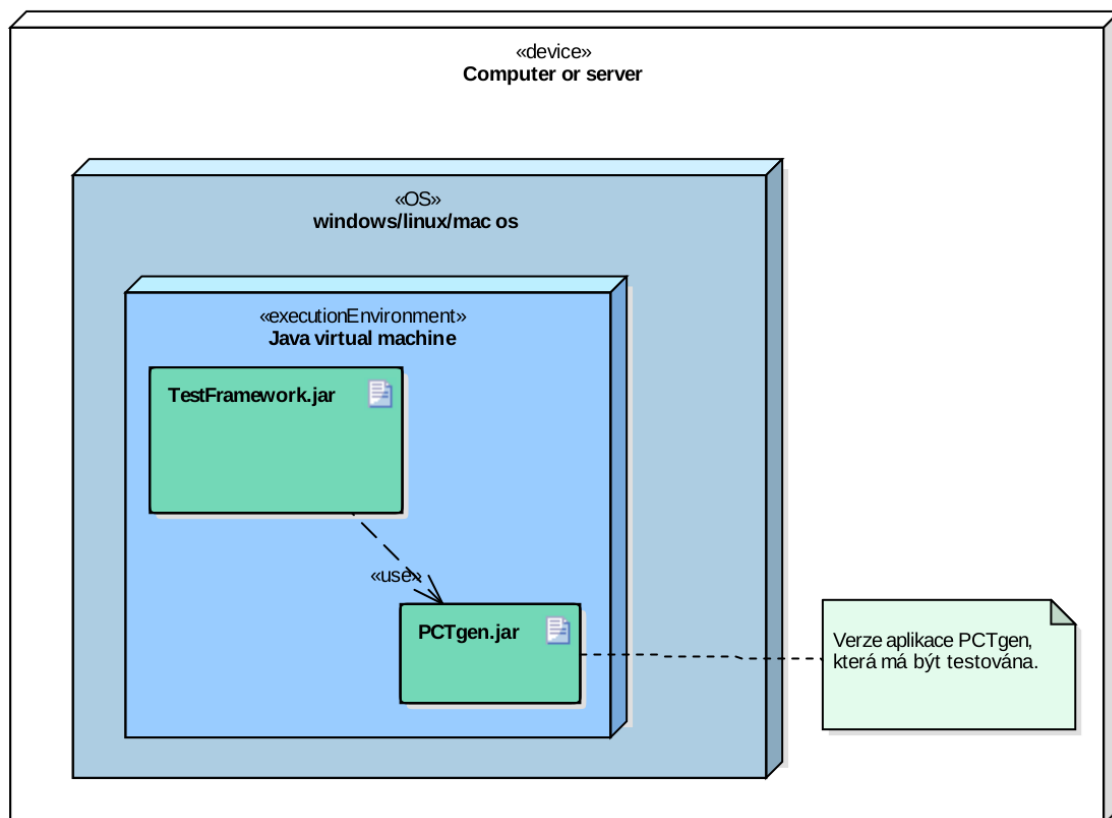
Metoda najde a vrátí dialogové okno ve kterém jsou vygenerovány testovací situace.

■ 5.4.7 TextConstants

Tato třída slouží k definování textových konstant v aplikaci. Framework vyhledává komponenty také podle textu, který je uložen v této třídě.

■ 5.5 Diagram nasazení

V této kapitole popíšeme a graficky znázorníme jakým způsobem je možné framework nainstalovat a spouštět.



Obrázek 5.1. Diagram nasazení.

Framework je aplikace naprogramovaná v programovacím jazyce Java 8, a tudíž je ke spuštění potřeba mít na počítači nebo serveru nainstalovanou Javu 8. Java je velmi rozšířenou platformou, a proto lze tuto aplikaci spouštět na většině dnes dostupných operačních systémech, jako je například Windows, Linux a Mac OS. Počítač či server může být cokoli, co podporuje výše zmíněný operační systém s podporou Javy. Hlavní výhodou tohoto frameworku a zároveň jeho předpokladem je spuštění aplikace na počítači, na kterém je vyvíjena aplikace PCTgen a nebo na vzdáleném serveru, který automatizovaně spouští regresní testy.

5.6 Spuštění testů

Do aplikace do složky /lib vložíme nejnovější spustitelný jar soubor aplikace PCTgen. Dále lze testy spustit dvěma způsoby. Za prvé z vývojového prostředí například z aplikace Netbeans nebo Intelij. Nebo za pomoci technologie maven z příkazové řádky, kde otevřeme složku, která obsahuje soubor pom.xml a příkazem :

```
mvn test
```

spustíme všechny testy v projektu.

Kapitola 6

Vytvořené testovací scénáře

Podle zadání a funkčních požadavků jsem vytvořil a naimplementoval testovací scénáře za pomoci tohoto testovacího frameworku. Následující testovací scénáře budou demonstrovat, jakým způsobem lze vytvářet a implementovat automatizované uživatelské testy pro aplikaci PCTgen. Jednotlivé testovací scénáře v této kapitole budou vždy samostatnou podkapitolou, kde název podkapitoly bude název testovacího scénáře.

6.1 Vytvořit 30 nových projektů

Popis:

Testovací scénář vytvoří 30 nových projektů a ověří názvy těchto vytvořených projektů. Každý nově vytvořený projekt by se měl jmenovat *New project N*, kde N je pořadové číslo projektu, v tomto případě 1 - 30.

Kroky testovacího scénáře:

1. Kliknout na tlačítko File.
2. 30x vybrat položku *New project*.
3. Ověřit, že se projekty jmenují podle specifikace PCTgen.

6.2 Vytvořit 30 nových grafů v jednom projektu

Popis:

Testovací scénář vytvoří 30 nových grafů a ověří názvy těchto vytvořených grafů. Každý nově vytvořený graf by se měl jmenovat *new graph N*, kde N je pořadové číslo grafu v projektu, v tomto případě 1 - 30.

Kroky testovacího scénáře:

1. Kliknout na tlačítko File.
2. Vybrat položku *New project*.
3. Rozbalit nově vytvořený projekt v levé části gui.
4. 30x opakovat vytvoření nového grafu v projektu -> pravým tlačítkem myši kliknout na projekt a vybrat položku *new graf*.
5. Ověřit, že se projekty jmenují podle specifikace PCTgen.

6.3 Vytvořit 30 nových vrcholů v jednom grafu.

Popis:

Testovací scénář vytvoří nový projekt, v novém projektu vytvoří nový graf a do grafu vloží 30 nových vrcholů. Testovací scénář má kontrolovat názvy nových vrcholů.

Kroky testovacího scénáře:

1. Kliknout na tlačítko File.
2. Vybrat položku *New project*.
3. Rozbalit nově vytvořený projekt v levé části gui.
4. Pravým tlačítkem myši kliknout na projekt a vybrat položku *new graf*.
5. Levým tlačítkem myši kliknout na nově vytvořený graf s názvem *new graph 1*.
6. Přesunout vrchol *start* z horní paletové komponenty do plátna uprostřed gui.
7. 30x přesunout vrchol *node* z horní paletové komponenty do plátna uprostřed gui.
8. Ověřit názvy vytvořených vrcholů podle PCTgen specifikace.

6.4 Vytvořit hrany střídavě v různých grafech

Popis:

Testovací scénář vytvoří dva různé grafy v jednom projektu a střídavě v těchto grafech vytvoří vrcholy, které pospojuje hranami a ověří, že automaticky vygenerované názvy se vytváří nezávisle na ostatních grafech. Jinak řečeno v prvním grafu se vytvoří nová hrana s názvem *1* a v druhém grafu se vytvoří hrana s názvem *1*.

Kroky testovacího scénáře:

1. Kliknout na tlačítko File.
2. Vybrat položku *New project*.
3. Rozbalit nově vytvořený projekt v levé části gui.
4. Pravým tlačítkem myši kliknout na projekt a vybrat položku *new graf*.
5. Pravým tlačítkem myši kliknout na projekt a vybrat položku *new graf*.
6. Levým tlačítkem myši kliknout na nově vytvořený graf s názvem *new graph 1*.
7. Přesunout vrchol *start* z horní paletové komponenty do plátna uprostřed gui.
8. Přesunout vrchol *node* z horní paletové komponenty do plátna uprostřed gui.
9. Vytvořit hranu z vrcholu *start* do vrcholu s názvem *A*.
10. Levým tlačítkem myši kliknout na nově vytvořený graf s názvem *new graph 2*.
11. Přesunout vrchol *start* z horní paletové komponenty do plátna uprostřed gui.
12. Přesunout vrchol *node* z horní paletové komponenty do plátna uprostřed gui.
13. Vytvořit hranu z vrcholu *start* do vrcholu s názvem *A*.
14. Ověřit, že v obou grafech se jediná hrana jmenuje *1*.

6.5 Vytvořit vrcholy střídavě v různých grafech

Popis:

Testovací scénář vytvoří dva různé grafy v jednom projektu a střídavě v těchto grafech vytvoří vrcholy a ověří, že automaticky vygenerované názvy se vytváří nezávisle na ostatních grafech. Tzv v prvním grafu se vytvoří nový vrchol s názvem *A* a v druhém grafu se vytvoří vrchol s názvem *A*.

Kroky testovacího scénáře:

1. Kliknout na tlačítko File.
2. Vybrat položku *New project*.
3. Rozbalit nově vytvořený projekt v levé části gui.
4. Pravým tlačítkem myši kliknout na projekt a vybrat položku *new graf*.
5. Pravým tlačítkem myši kliknout na projekt a vybrat položku *new graf*.
6. Levým tlačítkem myši kliknout na nově vytvořený graf s názvem *new graph 1*.
7. Přesunout vrchol *start* z horní paletové komponenty do plátna uprostřed gui.
8. Přesunout vrchol *node* z horní paletové komponenty do plátna uprostřed gui.
9. Levým tlačítkem myši kliknout na nově vytvořený graf s názvem *new graph 2*.
10. Přesunout vrchol *start* z horní paletové komponenty do plátna uprostřed gui.
11. Přesunout vrchol *node* z horní paletové komponenty do plátna uprostřed gui.
12. Ověřit, že v obou grafech jsou dvě hrany s názvy *start* a *A*.

6.6 Vytvořit grafy střídavě v různých projektech

Popis:

Testovací scénář ověří, zda automaticky vygenerované názvy grafů jsou generovány unikátně v projektu. Vytvoří se dva projekty a v každém projektu nový graf. Grafy v těchto různých projektech budou mít stejný název.

Kroky testovacího scénáře:

1. Kliknout na tlačítko File.
2. Vybrat položku *New project*.
3. Kliknout na tlačítko File.
4. Vybrat položku *New project*.
5. Rozbalit nově vytvořený projekt v levé části gui.
6. Pravým tlačítkem myši kliknout na projekt s názvem *new project 1* a vybrat položku *new graf*.
7. Pravým tlačítkem myši kliknout na projekt s názvem *new project 2* a vybrat položku *new graf*.
8. Ověřit, že graf v projektu 1 se nazývá *nový orientovaný graf 1*.
9. Ověřit, že graf v projektu 2 se nazývá *nový orientovaný graf 1*.

6.7 Vygenerování testovacích scénářů z grafu bez hran a vrcholů

Popis:

Testovací scénář ověří , že nelze vygenerovat testovací scénáře z grafu, který neobsahuje žádný jiný vrchol než *start*.

Kroky testovacího scénáře:

1. Kliknout na tlačítko File.
2. Vybrat položku *New project*.
3. Rozbalit nově vytvořený projekt v levé části gui.
4. Pravým tlačítkem myši kliknout na projekt s názvem *new project 1* a vybrat položku *new graf*.
5. Levým tlačítkem myši kliknout na nově vytvořený graf s názvem *new graph 1*.
6. Přesunout vrchol *start* z horní paletové komponenty do plátna uprostřed gui.
7. Kliknout na tlačítko *Generate test cases*.
8. Ověřit, zda aplikace zobrazila dialogové okno s popisem, že zde musí být alespoň jeden jiný vrchol než *start*.

6.8 Vygenerování testovacích scénářů z grafu bez vrcholu start

Popis:

Testovací scénář ověří , že nelze vygenerovat testovací scénáře z grafu, který neobsahuje vrchol *start*.

Kroky testovacího scénáře:

1. Kliknout na tlačítko File.
2. Vybrat položku *New project*.
3. Rozbalit nově vytvořený projekt v levé části gui.
4. Pravým tlačítkem myši kliknout na projekt s názvem *new project 1* a vybrat položku *new graf*.
5. Levým tlačítkem myši kliknout na nově vytvořený graf s názvem *new graph 1*.
6. Kliknout na tlačítko *Generate test cases*.
7. Ověřit, zda aplikace zobrazila dialogové okno s popisem, že v grafu musí být *start*.

6.9 Vygenerování testovacích scénářů z grafu bez vrcholu start

Název: *Vygenerování testovacích scénářů z grafu kde vrchol start má příchozí hranu.*

Popis:

Testovací scénář ověří , že nelze vygenerovat testovací scénáře z grafu, kde vrchol *start* má příchozí hranu.

Kroky testovacího scénáře:

1. Kliknout na tlačítko File.
2. Vybrat položku *New project*.
3. Rozbalit nově vytvořený projekt v levé části gui.
4. Pravým tlačítkem myši kliknout na projekt s názvem *new project 1* a vybrat položku *new graf*.
5. Levým tlačítkem myši kliknout na nově vytvořený graf s názvem *new graph 1*.
6. Přesunout vrchol *start* z horní paletové komponenty do plátna uprostřed gui.
7. Přesunout vrchol *node* z horní paletové komponenty do plátna uprostřed gui.
8. Spojit vrchol s názvem *A* s vrcholem *start* směrem od *A* do vrcholu *start*.
9. Kliknout na tlačítko *Generate test cases*.
10. Ověřit, zda aplikace zobrazila dialogové okno s popisem, že start nemůže mít příchozí hranu.

6.10 Každý vrchol v grafu musí mít příchozí hranu

Popis:

Testovací scénář ověří , že nelze generovat testovací scénáře z grafu, pokud existuje v grafu vrchol, který nemá příchozí hranu.

Kroky testovacího scénáře:

1. Kliknout na tlačítko File.
2. Vybrat položku *New project*.
3. Rozbalit nově vytvořený projekt v levé části gui.
4. Pravým tlačítkem myši kliknout na projekt s názvem *new project 1* a vybrat položku *new graf*.
5. Levým tlačítkem myši kliknout na nově vytvořený graf s názvem *new graph 1*.
6. Přesunout vrchol *start* z horní paletové komponenty do plátna uprostřed gui.
7. Přesunout vrchol *node* z horní paletové komponenty do plátna uprostřed gui.
8. Přesunout další vrchol *node* z horní paletové komponenty do plátna uprostřed gui.
9. Spojit vrchol s názvem *A* s vrcholem *start* směrem od *start* do vrcholu *A*.
10. Kliknout na tlačítko *Generate test cases*.
11. Ověřit, zda aplikace zobrazila dialogové okno s popisem, že vrchol nemá příchozí hranu.

6.11 Graf bez koncového vrcholu

Popis:

Testovací scénář ověří, že nelze generovat testovací scénáře z grafu, pokud neexistuje koncový vrchol v grafu.

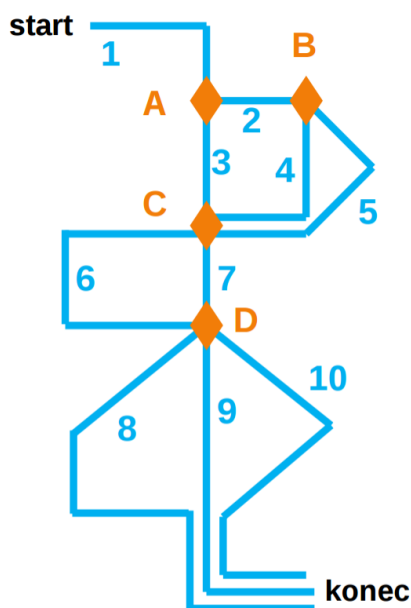
Kroky testovacího scénáře:

1. Kliknout na tlačítko File.
2. Vybrat položku *New project*.
3. Rozbalit nově vytvořený projekt v levé části gui.
4. Pravým tlačítkem myši kliknout na projekt s názvem *new project 1* a vybrat položku *new graf*.
5. Levým tlačítkem myši kliknout na nově vytvořený graf s názvem *new graph 1*.
6. Přesunout vrchol *start* z horní paletové komponenty do plátna uprostřed gui.
7. Přesunout vrchol *node* z horní paletové komponenty do plátna uprostřed gui.
8. Přesunout vrchol *node* z horní paletové komponenty do plátna uprostřed gui.
9. Spojit vrchol s názvem *A* s vrcholem *start* směrem od *start* do vrcholu *A*.
10. Spojit vrchol s názvem *B* s vrcholem *A* směrem od *A* do vrcholu *B*.
11. Spojit vrchol s názvem *B* s vrcholem *A* směrem od *B* do vrcholu *A*.
12. Kliknout na tlačítko *Generate test cases*.
13. Ověřit, zda aplikace zobrazila dialogové okno s popisem, že graf musí obsahovat koncový vrchol.

6.12 Vytvoření grafu a ověření vygenerovaných testovacích scénářů

Popis:

Testovací scénář vytvoří graf podle obrázku níže a ověří vygenerované testovací scénáře.



Obrázek 6.1. Ukázkový graf.¹⁾

Kroky testovacího scénáře:

1. Kliknout na tlačítko File.
2. Vybrat položku *New project*.
3. Rozbalit nově vytvořený projekt v levé části gui.
4. Pravým tlačítkem myši kliknout na projekt s názvem *new project 1* a vybrat položku *new graf*.
5. Levým tlačítkem myši kliknout na nově vytvořený graf s názvem *new graph 1*.
6. Přesunout vrchol *start* z horní paletové komponenty do plátna uprostřed gui.
7. Přesunout vrchol *node* z horní paletové komponenty do plátna uprostřed gui.
8. Přesunout vrchol *node* z horní paletové komponenty do plátna uprostřed gui.
9. Přesunout vrchol *node* z horní paletové komponenty do plátna uprostřed gui.
10. Přesunout vrchol *node* z horní paletové komponenty do plátna uprostřed gui.
11. Přesunout vrchol *node* z horní paletové komponenty do plátna uprostřed gui.
12. Spojit vrchol s názvem *A* s vrcholem *start* směrem od *start* do vrcholu *A*.
13. Spojit vrchol s názvem *A* s vrcholem *B* směrem od *A* do vrcholu *B*.
14. Spojit vrchol s názvem *A* s vrcholem *C* směrem od *A* do vrcholu *C*.
15. Spojit vrchol s názvem *B* s vrcholem *C* směrem od *B* do vrcholu *C*.
16. Spojit vrchol s názvem *B* s vrcholem *C* směrem od *B* do vrcholu *C*.
17. Spojit vrchol s názvem *C* s vrcholem *D* směrem od *C* do vrcholu *D*.

¹⁾ Obrázek převzat z přednášek Ing. Miroslav Bureš Ph.D. pro předmět Testování softwaru.

18. Spojit vrchol s názvem *C* s vrcholem *D* směrem od *C* do vrcholu *D*.
19. Spojit vrchol s názvem *D* s vrcholem *E* směrem od *D* do vrcholu *E*.
20. Spojit vrchol s názvem *D* s vrcholem *E* směrem od *D* do vrcholu *E*.
21. Spojit vrchol s názvem *D* s vrcholem *E* směrem od *D* do vrcholu *E*.
22. Kliknout na tlačítko *Generate test cases*.
23. Zapsat *1* do vstupního pole *Test depth level (TDL)*.
24. Potvrdit vygenerování testovacích scénářů.
25. Ověřit, že aplikace zobrazila nové okno s názvem *Test situations 1, TDL=1, ALG=PCT*.
26. Kliknout na záložku s názvem *Test situations*.
27. Ověřit, že tabulka testovacích scénářů obsahuje tuto kombinaci hran : 1 - 2 - 4 - 6 - 8 .
28. Ověřit, že tabulka testovacích scénářů obsahuje tuto kombinaci hran : 1 - 3 - 7 - 9.
29. Ověřit, že tabulka testovacích scénářů obsahuje tuto kombinaci hran : 1 - 2 - 5 - 6 - 10 .

6.13 Validace grafu pomocí tlačítka *validate graph*

Popis:

Testovací scénář ověří, tlačítko **validate graph** zvaliduje graf a odhalí nekonzistenci v grafu.

Kroky testovacího scénáře:

1. Kliknout na tlačítko *File*.
2. Vybrat položku *New project*.
3. Rozbalit nově vytvořený projekt v levé části gui.
4. Pravým tlačítkem myši kliknout na projekt s názvem *new project 1* a vybrat položku *new graf*.
5. Levým tlačítkem myši kliknout na nově vytvořený graf s názvem *new graph 1*.
6. Přesunout vrchol *start* z horní paletové komponenty do plátna uprostřed gui.
7. Kliknout na tlačítko *Validate graph*.
8. Ověřit, zda aplikace zobrazila dialogové okno s popisem, že zde musí být alespoň jeden jiný vrchol než *start*.

6.14 Validace dialogu pro generování testovacích scénářů

Popis:

Testovací scénář ověří, že dialogové okno s nastavením, podle kterého se následně generují testovací scénáře, je odolný vůči nevalidním vstupům.

Kroky testovacího scénáře:

1. Kliknout na tlačítko *File*.
2. Vybrat položku *New project*.

3. Rozbalit nově vytvořený projekt v levé části gui.
4. Pravým tlačítkem myši kliknout na projekt s názvem *new project 1* a vybrat položku *new graf*.
5. Levým tlačítkem myši kliknout na nově vytvořený graf s názvem *new graph 1*.
6. Přesunout vrchol *start* z horní paletové komponenty do plátna uprostřed gui.
7. Přesunout vrchol *node* z horní paletové komponenty do plátna uprostřed gui.
8. Spojit vrchol s názvem *A* s vrcholem *start* směrem od *start* do vrcholu *A*.
9. Kliknout na tlačítko vygenerovat testovací scénáře.
10. Do vstupního pole *Test depth level (TDL)* se zapíše *a*.
11. Potvrdí se vygenerování testovacích scénářů.
12. Ověří se, že aplikace zobrazila dialogové okno, kde upozorňuje na špatný vstup.
13. Do vstupního pole *Test depth level (TDL)* se zapíše *0*.
14. Potvrdí se vygenerování testovacích scénářů.
15. Ověří se, že aplikace zobrazila dialogové okno, kde upozorňuje na špatný vstup.
16. Do vstupního pole *Test depth level (TDL)* se zapíše *-1*.
17. Potvrdit vygenerování testovacích scénářů.
18. Ověří se, že aplikace zobrazila dialogové okno, kde upozorňuje na špatný vstup.

6.15 Očekávané oznámení o neuložení projektu

Popis:

Testovací scénář ověří, že při neuložení projektu, aplikace na tuto skutečnost upozorní dialogovým oknem.

Kroky testovacího scénáře:

1. Kliknout na tlačítko File.
2. Vybrat položku *New project*.
3. Rozbalit nově vytvořený projekt v levé části gui.
4. Pravým tlačítkem myši kliknout na projekt s názvem *new project 1* a vybrat položku *new graf*.
5. Zavřít aplikaci.
6. Zkontrolovat, že aplikace zobrazila dialogové okno, že projekt není uložen.

6.16 Neočekávané oznámení o neuložení projektu.

Popis:

Testovací scénář ověří, že při uložení projektu, aplikace neupozorní na to, že projekt nebyl uložen.

Kroky testovacího scénáře:

1. Kliknout na tlačítko File.
2. Vybrat položku *New project*.
3. Kliknout na tlačítko File.
4. Vybrat položku *uložit projekt*

5. Zavřít aplikaci.
6. Ověřit, zda se nezobrazilo dialogové okno, že projekt není uložen.

6.17 Defaultní crud model

Popis:

Testovací scénář ověří, že při vytvoření nového projektu, jsou v projektu již vytvořeny tyto objekty: CRUD matice, entity a funkce.

Kroky testovacího scénáře:

1. Kliknout na tlačítko File.
2. Vybrat položku *New project*.
3. Zkontrolovat, že v levém panelu je položka *CRUD matice*.
4. Zkontrolovat, že v levém panelu je položka *entity*.
5. Zkontrolovat, že v levém panelu je položka *funkce*.

6.18 Uložení projektu do souboru

Popis:

Testovací scénář ověří, že při uložení projektu se projekt opravdu uloží.

Kroky testovacího scénáře:

1. Kliknout na tlačítko File.
2. Vybrat položku *New project*.
3. Kliknout na tlačítko File.
4. Vybrat položku *uložit projekt*.
5. Zkontrolovat, zda je projekt uložen.

6.19 Uložení projektu a následné nahrání projektu

Popis:

Testovací scénář ověří, že lze projekt uložit a následně nahrát zpět do aplikace.

Kroky testovacího scénáře:

1. Kliknout na tlačítko File.
2. Vybrat položku *New project*.
3. Kliknout na tlačítko File.
4. Vybrat položku *save projekt*.
5. Zkontrolovat, zda je projekt uložen.
6. Pravým tlačítkem myši, kliknout na *New project 1*.
7. Vybrat položku *Close projekt*.

8. Potvrdit v dialogovém okně pomocí tlačítka *ok*.
9. Kliknout na tlačítko *File*.
10. Vybrat položku *Open project*.
11. Zkontrolovat, že projekt je náhrán v aplikaci.

6.20 Smoke test bez žádné priority u hran

Popis:

Testovací scénář ověří, že nelze vygenerovat tabulku s testovacími scénáři, pokud uživatel nevyplní u žádné hrany priority.

Kroky testovacího scénáře:

1. Kliknout na tlačítko *File*.
2. Vybrat položku *New project*.
3. Rozbalit nově vytvořený projekt v levé části gui.
4. Pravým tlačítkem myši kliknout na projekt s názvem *new project 1* a vybrat položku *new graf*.
5. Levým tlačítkem myši kliknout na nově vytvořený graf s názvem *new graph 1*.
6. Přesunout vrchol *start* z horní paletové komponenty do plátna uprostřed gui.
7. Přesunout vrchol *node* z horní paletové komponenty do plátna uprostřed gui.
8. Spojit vrchol s názvem *A* s vrcholem *start* směrem od *start* do vrcholu *A*.
9. Kliknout na tlačítko vygenerovat testovací scénáře.
10. Zapsat *1* do vstupního pole *Test depth level (TDL)*.
11. Vybrat algoritmus *smoke test*.
12. Potvrdit vygenerování testovacích scénářů.
13. Ověřit, že aplikace zobrazila dialogové okno s chybovou hláškou, že není u žádné hrany definována priorita.

6.21 Smoke test se stejnými prioritami na hranách.

Popis:

Testovací scénář ověří, že nelze vygenerovat tabulku s testovacími scénáři, pokud uživatel vyplní všechny priority u hran stejně.

Kroky testovacího scénáře:

1. Kliknout na tlačítko *File*.
2. Vybrat položku *New project*.
3. Rozbalit nově vytvořený projekt v levé části gui.
4. Pravým tlačítkem myši kliknout na projekt s názvem *new project 1* a vybrat položku *new graf*.
5. Levým tlačítkem myši kliknout na nově vytvořený graf s názvem *new graph 1*.
6. Přesunout vrchol *start* z horní paletové komponenty do plátna uprostřed gui.
7. Přesunout vrchol *node* z horní paletové komponenty do plátna uprostřed gui.
8. Přesunout vrchol *node* z horní paletové komponenty do plátna uprostřed gui.

9. Spojit vrchol s názvem *A* s vrcholem *start* směrem od *start* do vrcholu *A*.
10. Spojit vrchol s názvem *A* s vrcholem *B* směrem od *A* do vrcholu *B*.
11. Kliknout na hranu mezi vrcholem *A* a vrcholem *start*.
12. V pravém panelu vyplnit prioritu HIGH.
13. Kliknout na hranu mezi vrcholem *A* a vrcholem *B*.
14. V pravém panelu vyplnit prioritu HIGH.
15. Kliknout na tlačítko vygenerovat testovací scénáře.
16. Zapsat *1* do vstupního pole *Test depth level (TDL)*.
17. Vybrat algoritmus *smoke test*.
18. Potvrdit se vygenerování testovacích scénářů.
19. Ověřit, že aplikace zobrazila dialogové okno s chybovou hláškou, že všechny hrany mají stejnou prioritu.

6.22 Neaktuálních testovacích scénáře

Popis:

Testovací scénář ověří, že pokud uživatel vygeneruje testovací scénáře z určitého grafu, který následně upraví, tak ho aplikace na tuto skutečnost upozorní pomocí dialogového okna.

Kroky testovacího scénáře:

1. Kliknout na tlačítko File.
2. Vybrat položku *New project*.
3. Rozbalit nově vytvořený projekt v levé části gui.
4. Pravým tlačítkem myši kliknout na projekt s názvem *new project 1* a vybrat položku *new graf*.
5. Levým tlačítkem myši kliknout na nově vytvořený graf s názvem *new graph 1*.
6. Přesunout vrchol *start* z horní paletové komponenty do plátna uprostřed gui.
7. Přesunout vrchol *node* z horní paletové komponenty do plátna uprostřed gui.
8. Přesunout vrchol *node* z horní paletové komponenty do plátna uprostřed gui.
9. Spojit vrchol s názvem *A* s vrcholem *start* směrem od *start* do vrcholu *A*.
10. Spojit vrchol s názvem *A* s vrcholem *B* směrem od *A* do vrcholu *B*.
11. Kliknout na tlačítko vygenerovat testovací scénáře.
12. Zapsat *1* do vstupního pole *Test depth level (TDL)*.
13. Potvrdit vygenerování testovacích scénářů.
14. Zavřít okno s testovacími scénáři.
15. Přesunout vrchol *node* z horní paletové komponenty do plátna uprostřed gui.
16. Spojit vrchol s názvem *B* s vrcholem *C* směrem od *B* do vrcholu *C*.
17. V levém panelu levým tlačítkem myši kliknout na dříve vygenerované testovací scénáře.
18. Ověřit, že aplikace zobrazí dialogové okno s upozorněním, že scénáře již nemusí být aktuální.

6.23 Validace validního grafu

Popis:

Validace validního grafu by měla zobrazit dialogové okno s textem *No inconsistencies found. Test cases can be generated.*

Kroky testovacího scénáře:

1. Kliknout na tlačítko File.
2. Vybrat položku *New project*.
3. Rozbalit nově vytvořený projekt v levé části gui.
4. Pravým tlačítkem myši kliknout na projekt s názvem *new project 1* a vybrat položku *new graf*.
5. Levým tlačítkem myši kliknout na nově vytvořený graf s názvem *new graph 1*.
6. Přesunout vrchol *start* z horní paletové komponenty do plátna uprostřed gui.
7. Přesunout vrchol *node* z horní paletové komponenty do plátna uprostřed gui.
8. Přesunout vrchol *node* z horní paletové komponenty do plátna uprostřed gui.
9. Spojit vrchol s názvem *A* s vrcholem *start* směrem od *start* do vrcholu *A*.
10. Spojit vrchol s názvem *A* s vrcholem *B* směrem od *A* do vrcholu *B*.
11. Kliknout na tlačítko *validate graph*.
12. Ověřit, zda aplikace zobrazila dialogové okno s popisem, že nebyly nalezené žádné nekonzistence v grafu.

6.24 Smazání vrcholu v grafu

Popis:

Testovací scénář ověří, že lze vrchol smazat z grafu.

Kroky testovacího scénáře:

1. Kliknout na tlačítko File.
2. Vybrat položku *New project*.
3. Rozbalit nově vytvořený projekt v levé části gui.
4. Pravým tlačítkem myši kliknout na projekt s názvem *new project 1* a vybrat položku *new graf*.
5. Levým tlačítkem myši kliknout na nově vytvořený graf s názvem *new graph 1*.
6. Přesunout vrchol *start* z horní paletové komponenty do plátna uprostřed gui.
7. Přesunout vrchol *node* z horní paletové komponenty do plátna uprostřed gui.
8. Přesunout vrchol *node* z horní paletové komponenty do plátna uprostřed gui.
9. Spojit vrchol s názvem *A* s vrcholem *start* směrem od *start* do vrcholu *A*.
10. Spojit vrchol s názvem *A* s vrcholem *B* směrem od *A* do vrcholu *B*.
11. Smazat vrchol pomocí pravého kliku myši na vrchol s názvem *A*.
12. Ověřit, zda se zobrazil dialog s textem : *Really delete the graph item?*
13. Potvrdit smazání vrcholu pomocí tlačítka : *OK*.
14. Ověřit, že se vrchol smazal.

6.25 Start v grafu může být jen jednou

Popis:

Testovací scénář ověří, že lze vložit vrchol *start* do grafu jen jednou.

Kroky testovacího scénáře:

1. Kliknout na tlačítko File.
2. Vybrat položku *New project*.
3. Rozbalit nově vytvořený projekt v levé části gui.
4. Pravým tlačítkem myši kliknout na projekt s názvem *new project 1* a vybrat položku *new graf*.
5. Levým tlačítkem myši kliknout na nově vytvořený graf s názvem *new graph 1*.
6. Přesunout vrchol *start* z horní paletové komponenty do plátna uprostřed gui.
7. Přesunout vrchol *start* z horní paletové komponenty do plátna uprostřed gui.
8. Ověřit, že aplikace zobrazila dialogové okno s textem, že graf již obsahuje vrchol *start*.

Kapitola 7

Výsledky běhu vytvořených testů

Několik testovacích scénářů objevilo chyby v aplikaci PCTgen. Některé scénáře jsem již programoval s tím, že jsem chybu znal a tyto scénáře tedy mohou posloužit jako regresní testy při opravení chyby. Některé scénáře zase naopak objevili chybu, kterou by například běžný manuální tester neobjevil.

Tyto testovací scénáře objevili chybu :

Testovací scénář s názvem *Vytvořit 30 nových projektů.* :

Výsledek:

Po 24. projektu se projekty zase číslují od jedničky a aplikace spadne.

Testovací scénář s názvem *Vytvořit 30 nových grafů v jednom projektu :*

Výsledek:

Po 24. nově vytvořeným grafu se projekty zase číslují od jedničky a aplikace spadne.

Testovací scénář s názvem *Vytvořit 30 nových vrcholů v jednom grafu:*

Výsledek:

Po 24. nově vytvořeným vrcholu v grafu se začnou duplicitně jmenovat vrcholy v grafu, z kterého následně nelze vygenerovat testovací scénáře.

Testovací scénář s názvem *Vytvořit hrany střídavě v různých grafech.*

Výsledek:

Názvy vrcholů jsou vytvářeny v závislosti na předchozích názvech v jiných grafech.

Testovací scénář s názvem *Vytvořit vrcholy střídavě v různých grafech.*

Výsledek:

Názvy hran jsou vytvářeny v závislosti na předchozích názvech v jiných grafech.

Testovací scénář s názvem *Vytvořit grafy střídavě v různých projektech.*

Výsledek:

Vytváření grafů v různých projektech není nezávislé, ale názvy jsou vytvářeny v závislosti na předchozích názvech grafů i v jiných projektech.

Testovací scénář s názvem

Název: Neočekávané oznámení o neuložení projektu.

Výsledek:

Pokud se projekt uloží, aplikace při ukončení zobrazí dialogové okno, že projekt není uložen.

Testovací scénář č.14.

Název: *Validace dialogu pro generování testovacích scénářů.*

Výsledek:

Při vstupu 0 a -1 do textového pole pro TDL ¹⁾ v dialogovém okně na generování testovacích scénářů se nezobrazí chyba a dialogové okno zmizí a nic se nestane.

¹⁾ Test depth level

Kapitola 8

Testování frameworku

Po domluvě s vedoucím bakalářské práce s panem Ing. Miroslavem Burešem Ph.D je dostačující otestovat framework samotným vývojem 25-ti uživatelských testů Spuštění a běh vyvinutých testovacích scénářů proto dostatečně ověřuje funkčnost této práce. Dalším důležitým krokem bylo framework otestovat na více počítačích a na různých operačních systémech. Framework jsem vyvíjel na operačním systému Ubuntu 16.04 a zde jsem také framework odladil. Při testování na operačním systému Windows a na počítači s odlišným rozlišením monitoru jsem však odhalil chybu, která zamezovala automatizovanému kreslení grafů za pomoci frameworku. Tuto chybu jsem opravil a důkladně otestoval opakovaným spouštěním testů. Do budoucna by však bylo vhodné framework otestovat na více počítačích a operačních systémech. Kvůli omezeným možnostem jsem framework otestoval pouze na čtyřech počítačích s operačními systémy Ubuntu 16.04, Windows 10, macOS sierra a windows 8.1. Všechny testy jsem spouštěl skrze oficiální distribuci Javy.

Kapitola 9

Závěr

Zadáním bakalářské práce bylo vytvořit framework na snadné vytváření uživatelských testů a následně tento framework demonstrovat na 25-ti testovacích scénářích. V práci jsem tedy uvedl do problematiky testování softwaru a popsal konkrétně aspekty vývoje frameworku. Dále jsem zdokumentoval nejdůležitější části frameworku a popsal testovací scénáře, na kterých jsem demonstroval jeho funkčnost. Framework se mi podařilo vyvinout a demonstrovat jej na 25-ti testovacích scénářích. Tudíž se mi podařilo cíl práce naplnit a vývojáři aplikace PCTgen mohou v budoucnu tento framework používat na programování automatizovaných uživatelských testů. Některé testovací scénáře, které měly demonstrovat tuto práci, objevily chyby v aplikaci PCTgen. Dokonce jsem při implementaci tohoto frameworku narazil na špatně naimplementovaná vlákna v aplikaci. V průběhu vývoje tohoto frameworku však byla aplikace PCTgen aktivně vyvíjena a některé chyby by již mohly být opravené. Dále jsem se zpočátku potýkal s problémem, který vznikl proto, že jsem tuto práci vyvinul přímo v projektu aplikace PCTgen, avšak nakonec jsem musel tuto práci vyextrahovat a vytvořit jako samostatnou aplikaci, jelikož projekt PCTgen byl aktivně vyvíjen a vznikaly rozsáhlé konflikty ve zdrojových kódech. Z toho také plyne, že tento framework je vyvinut na neaktuální verzi aplikace a musel by se nejspíše upravit na aktuální verzi. Pokud by tedy v budoucím vývoji vývojáři projektu PCTgen chtěli používat tento framework na psaní uživatelských end-ToEnd testů, bylo by nejvýhodnější tento framework vložit přímo do zdrojových kódů aplikace PCTgen, kde by se tento framework mohl snadno udržovat, rozšiřovat a používat. Pak by například bylo možné používat nástroj na Continuous Integration ¹⁾, kdy vývojář vloží ²⁾ práci do repozitáře a tento nástroj spustí sadu testů a oznámí, zda testy proběhly v pořádku.

¹⁾ Nástroj na automatizované sestavení a spuštění testů po vložení práce do repozitáře.

²⁾ V gitu například pomocí příkazu push



Literatura

[1] B. W. Boehm, R. K. McClean and D. B. Urfrig. Some experience with automated aids to the design of large-scale reliable software. *IEEE Transactions on Software Engineering* SE-1(1), pp. 125-133. 1975. . DOI: 10.1109/TSE.1975.6312826.

[2] JUnit [online]. [cit. 2016-12-30]. Dostupné z: <http://junit.org/junit4/>

[3] Kent Beck, *Test Driven Development: By Example*. Addison-Wesley, 2003

[4] M. Bureš. PCTgen: Automated Generation of Test Cases for Application Workflows. In: *New Contributions in Information Systems and Technologies, Advances in Intelligent Systems and Computing*. vol.353, Springer, 2015, s. 789-794.

[5] <http://docs.oracle.com/javase/6/docs/api/Javax/swing/package-summary.html>

[6] Apache Maven Project [online]. [cit. 2016-12-30]. Dostupné z: <https://maven.apache.org/>

[7] BUILD INTERACTIVE WEB DIAGRAMMING APPS [online]. [cit. 2016-12-30]. Dostupné z: <https://www.jgraph.com/index.html>

[8] Sikuli script [online]. [cit. 2016-12-30]. Dostupné z: <http://www.sikuli.org/>

[9] AssertJ Swing: Features of AssertJ Swing [online]. [cit. 2016-12-30]. Dostupné z: <http://joel-costigliola.github.io/assertj/assertj-swing.html>

[10] Přednášky Ing. Miroslav Bureš Ph.D. pro předmět Testování softwaru.

[11] SeleniumHQ Browser Automation [online]. [cit. 2017-01-05]. Dostupné z: <http://www.seleniumhq.org/>

[12] TestNG [online]. [cit. 2017-01-05]. Dostupné z: <http://testng.org/doc/index.html>

[13] Gradle [online]. [cit. 2017-01-05]. Dostupné z: [13] <https://gradle.org/>

[14] BLOCH, Joshua. *Effective Java*. 2nd ed. Upper Saddle River, N.J.: Addison-Wesley, c2008. Java Series. ISBN 978-0-321-35668-0.

[15] SCHILDT, Herbert. *Java: the complete reference*. Ninth edition. New York: McGraw-Hill Education, 2014. ISBN 9780071808552.

- [16] HUNT, Andrew a David THOMAS. Pragmatic unit testing in Java with JUnit. Raleigh, N.C.: Pragmatic Bookshelf, c2003. ISBN 0974514012.
- [17] COHEN, Frank. Java testing and design: from unit testing to automated Web tests. Upper Saddle River, NJ: Prentice Hall PTR, 2004. ISBN 0131421891.
- [18] PATTON, Ron. Software testing. 2nd ed. Indianapolis, IN: Sams Pub., c2006. ISBN 0-672-32798-8.
- [19] BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ. Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu. Praha: Grada, 2016. Profesionál. ISBN 978-80-247-5594-6.



Příloha **A**

Obsah přiloženého cd

- Zdrojový kód frameworku.
- Ve složce /lib jar soubor aplikace PCTgen.
- Javadoc dokumentace frameworku.
- tex soubor bakalářské práce.
- pdf soubor bakalářské práce.

Příloha B

Požadavky na používání frameworku

Pro používání frameworku stačí mít v počítači nainstalované JRE 8 ¹⁾ prostředí. Pokud by uživatel chtěl spouštět testy za pomoci technologie maven, musí si maven nainstalovat, to lze na všechny běžně používané operační systémy.

¹⁾ Java SE Runtime Environment