Mendelova univerzita v Brně
Provozně ekonomická fakulta

Automatizované testy bankovního softwaru

Diplomová práce

Vedoucí práce:
Ing. Naděžda Chalupová, Ph.D.

Bc. Michal Gajdošík

Brno 2015

Tímto bych rád poděkoval Ing. Naděždě Chalupové, Ph.D. za cenné rady, připomínky a konzultace. Dále děkuji své přítelkyni a rodině za podporu při psaní této práce i v průběhu celého studia. V neposlední řadě bych rád poděkoval doc. Ing. Dr. Jiřímu Rybičkovi za sázecí styl systému LaTeX pro závěrečné práce.

Čestné prohlášení

Prohlašuji, že jsem tuto práci: **Automatizované testy bankovního softwaru** vypracoval samostatně a veškeré použité prameny a informace jsou uvedeny v seznamu použité literatury. Souhlasím, aby moje práce byla zveřejněna v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů, a v souladu s platnou *Směrnicí o zveřejňování vysokoškolských závěrečných prací.*
Jsem si vědom, že se na moji práci vztahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brně má právo na uzavření licenční smlouvy a užití této práce jako školního díla podle § 60 odst. 1 Autorského zákona.
Dále se zavazuji, že před sepsáním licenční smlouvy o využití díla jinou osobou (subjektem) si vyžádám písemné stanovisko univerzity o tom, že předmětná licenční smlouva není v rozporu s oprávněnými zájmy univerzity, a zavazuji se uhradit případný příspěvek na úhradu nákladů spojených se vznikem díla, a to až do jejich skutečné výše.

V Brně dne 18. května 2015 .....

Abstract

Gajdošík, M. Automated tests of banking software. Master Thesis. Brno, 2015.

The objective of this work is to design and implement a set of automated tests to verify the correct functioning of banking software features. The work introduces tools for automated testing. The output of work is a set of automated tests, which are created by using tools including SoapUI, IntelliJ IDEA and PL/SQL Developer. The benefit of this work is efficient and repeatable control of the required functionality and to create a design that can be used for creating automated tests.

Keywords

Automation, Groovy, SoapUI, testing, WSDL

Abstrakt

Gajdošík, M. Automatizované testy bankovního softwaru. Diplomová práce. Brno, 2015.

Cílem této práce je návrh a implementace sady automatizovaných testů pro testování funkčnosti bankovního softwaru. V práci jsou představeny nástroje určené pro automatizované testování. Výstupem práce je sada automatizovaných testů, jenž byly vytvořeny pomocí nástrojů SoapUI, IntelliJ IDEA a PL/SQL Developer. Přínosem této práce je především efektivní a opakovatelná kontrola požadované funkčnosti a vytvoření návrhu, pomocí kterého lze postupovat při tvorbě automatizovaných testů.

Klíčová slova

Automatizace, Groovy, SoapUI, testování, WSDL

Obsah

1 Úvod a cíl práce

1.1 Úvod

Kontrola kvality je důležitou a nedílnou součástí výrobního a vývojového procesu. Pokud by se zanedbávala nebo vůbec neprováděla, v určitých oblastech by mohla mít absence tohoto úkonu nedozírné následky. Z tohoto důvodu je potřebné provádět důslednou kontrolu a to zejména u kritických částí procesu. Při provádění manuální kontroly se některá oblast může snadno přehlédnout nebo se v rámci únavy neprovede její kontrola tak detailně. Jako řešení se nabízí automatizace této činnosti.

Automatizace činností je oblast, která se neustále rozvíjí a zdokonaluje. Pomocí automatizace lze dosáhnout efektivnějších, rychlejších a přesnějších výsledků než při provádění stejné činnosti manuálně. Existuje mnoho druhů automatizace, ať už se jedná o automatizaci výrobních procesů, kontroly kvality apod. Proces automatizace je v dnešní době velmi rozšířen a používá se v širokém spektru oblastí.

V práci bude přiblížena problematika testování softwaru. Dále budou představeny nástroje pro správu a řízení testů, nástroje umožňující automatizované a výkonnostní testování. V poslední kapitole budou definovány požadavky pro automatizaci, proveden návrh a následně budou automatizované testy implementovány.

1.2 Cíl práce

Cílem práce bude navrhnout a implementovat sadu automatizovaných testů pro testování funkčnosti bankovního softwaru. Automatizované testy budou vytvořeny pro specifickou webovou službu, která patří mezi kritickou část systému.

2 Testování softwaru

V ideálním světě by bylo nejlepší možnou variantou otestovat veškeré možné kombinace, které v testované aplikaci mohou nastat. V některých případech to však není možné. I jednoduchý program může mít tisíce různých kombinací. Vytvoření testovacích případů pro každý jednotlivý průchod je nepraktické a zabere mnoho času a lidských zdrojů (Myers, 2012).

Pro nasimulování všech kombinací se jeví jako nejlepší řešení požadované průchody zautomatizovat. Ne vždy je ale možné automatizaci provést. Někdy může být překážkou obtížnost vytvoření požadovaných testů, případně se v konečném důsledku nevyplatí investice do implementace automatizovaných testů. Problematiku testování lze definovat a popsat několika způsoby:

- Testování může být velmi efektivním způsobem, jak dokázat přítomnost chyb, ale je beznadějně nevhodné k prokázání jejich nepřítomnosti (Dijkstra, 1972).
- Proces získávání důvěry v to, že program nebo systém dělá, co se od něj očekává (Hetzel, 1973).
- Kvalita je zdarma, ale jen pro ty, kteří jsou ochotni za ni těžce zaplatit (DeMarco, 2012).
- Lidé pod tlakem nepracují lépe, pouze pracují rychleji (DeMarco, 2012).
- Systematické prozkoumávání programu nebo systému, jehož hlavním záměrem je hledání a reportování chyb (Morgan, Samaroo a Hambling, 2009).
- Provozování programu nebo systému za účelem hledání chyb (Myers, 2012).

V následujících podkapitolách budou přiblíženy elementární informace v oblasti testování.

2.1 Funkční a nefunkční testování

Funkční testování

Funkční testování je zaměřeno na funkční požadavky, které jsou zadány od zákazníka. Jedná se o konkrétní funkčnost aplikace. Tyto požadavky jsou následně zpracovávány analytiky. Ke zpracování požadavků je vhodné použít vhodný nástroj, například Enterprise Architect. Ten umožňuje komplexní modelování funkcionality, vyhledávání dle klíčových slov, jednoznačného identifikátoru GUID apod. Následně jsou tyto požadavky testovány manuálně nebo jsou automatizovány.

Nefunkční testování

Nefunkční testování je zaměřeno na nefunkční požadavky, které jsou podstatné pro korektní fungování. Mohou to být například požadavky na výkon, velikost, spo-

lehlivost, dostupnosti apod. Mezi nefunkční testy lze zařadit především výkonové, zátěžové a penetrační testy. Výkonové a zátěžové testy ověřují, jak testovaný produkt pracuje pod velkou zátěží. Penetrační testy testují bezpečnost testovaného softwaru.

2.2 Metodiky vývoje softwaru

V současné době existuje velké množství nejrůznějších metodik. Zvolení správného postupu je základním pilířem při vývoji softwaru. Někdy může firma používat určitou metodiku, ale s postupem času se může dojít ke zjištění, že použití jiného přístupu by mohlo být mnohem efektivnější. Ne vždy je ale tento proces možný. Pokud se firma rozhodne pro přechod, nestane se tak okamžitě. Většinou je potřeba veškeré zaměstnance zaškolit do nově zvolené metodiky, někteří s přechodem nemusí souhlasit apod.

Základem vývoje softwaru je tedy bezesporu vhodně zvolená metodika. Ta reprezentuje šablonu, která mapuje práva a zodpovědnosti každého jednotlivého účastníka při vývoji. Každý by tedy měl být obeznámen s tím, v jaké fázi nastává jeho role a jaký je očekávaný výsledek jeho práce.

V současné době existuje několik druhů nejrůznějších metodik. Několik z nich bude přiblíženo na následujících řádcích.

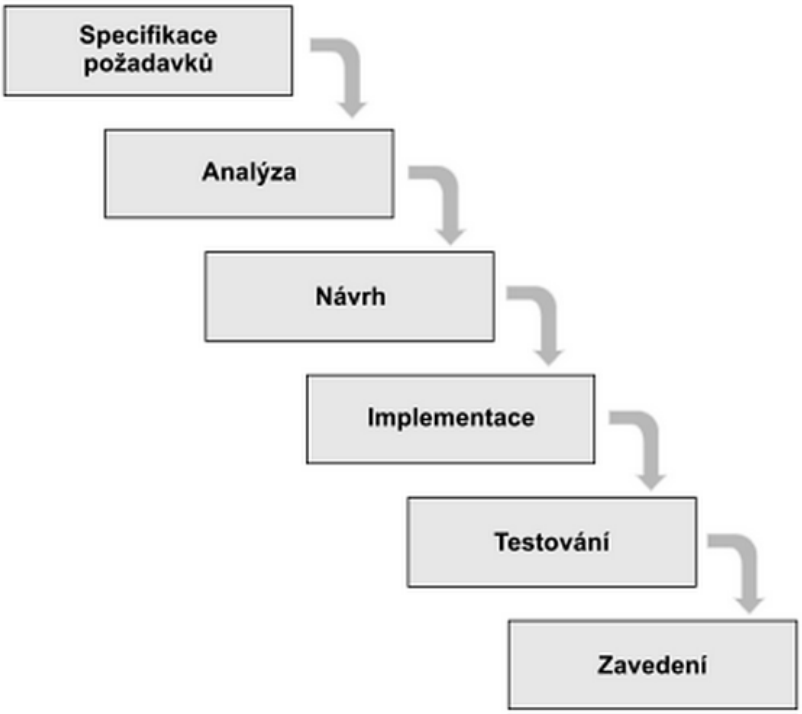
Vodopádový model

Vodopádový model je historicky jedním z nejstarších modelů pro vývoj softwaru. V dnešní době se většinou používají některé jeho modifikace, například iterativní přístup – jedná se o vodopádový model, který je rozdělen do jednotlivých iterací.

Vodopádový model představoval v době svého vzniku významný pokrok. Rozdělil totiž proces vývoje do samostatně oddělených fází. Tím pádem umožnil proces vývoje, který byl systematický a opakovatelný. Vodopádový model je vhodný pro situace, kdy je možné pokrýt a definovat všechny požadavky na finální produkt. Zároveň je žádoucí, aby se požadavky během vývoje příliš neměnily. V takovém případě lze očekávat dobré výsledky a dokáže poskytnout představu o rozsahu řešení. Na druhou stranu, pokud není možné specifikovat všechny požadavky již na začátku projektu, případně se požadavky mění v průběhu vývoje, je vhodné použít jinou metodiku. Další nevýhodou je i skutečnost, že zákazník je do procesu vývoje zapojen na začátku a na konci. V průběhu projektu tedy nemá dostatečnou kontrolu. Jako největší nevýhodou se jeví pozdní integrace systému, která se provádí až po implementaci všech požadovaných částí. Není výjimkou, že po tak obsáhlé integraci se mohou objevit problémy, které mohou vyžadovat změnu návrhu, následnou úpravu v kódu, což má za následek zpoždění celého projektu (Bruckner, 2012).

Iterativní přístup

Iterativní přístup je modifikovaná verze vodopádového modelu. Narozdíl od vodopádového modelu jsou případné změny ve specifikaci požadavků v průběhu projektu



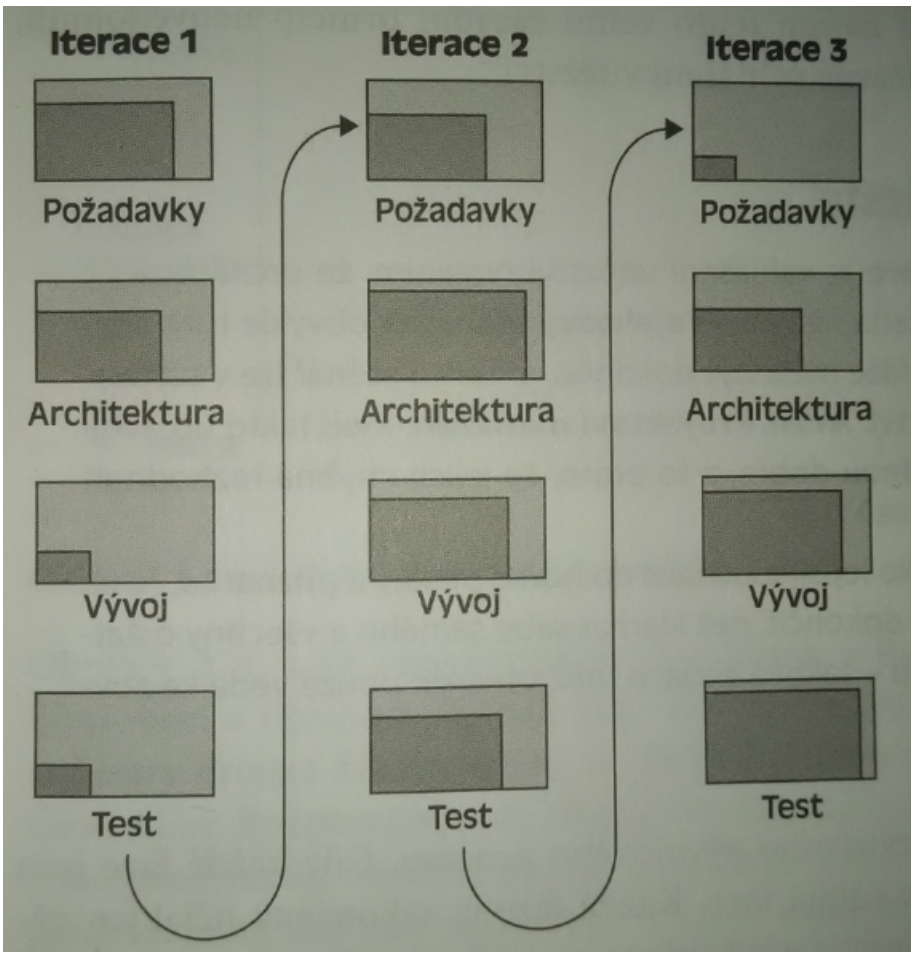
Obrázek 1: Vodopádový model (Bruckner, 2012)

lépe realizovatelné. Během jedné iterace jsou postupně provedeny všechny fáze projektu.

V každé jednotlivé iteraci je určena posloupnost jednotlivých činností a jejich časové omezení. Ideální situace je, pokud jsou v každé iteraci dokončeny veškeré úkoly. Iterace má určenou dobu trvání. V prvních iteracích se nejvíce času stráví zpravidla na požadavcích od zákazníka a následné analýze těchto požadavků. Po analýze požadavků je vytvořena funkční specifikace. Funkční specifikace obsahuje veškerou novou a upravovanou stávající funkcionalitu. Na základě funkční specifikace se může provést vývoj a následné testování.

Jakmile jsou úspěšně dokončeny všechny iterace, nastává konečný finální test, který zahrnuje kontrolu funkcionalitu všech iterací. Poté již proběhne nasazení nové funkcionality na produkční prostředí k zákazníkovi.

Nejedná se ale zdaleka o všechny činnosti. V jednotlivých iteracích dochází k průběžným předávacím schůzkám se zákazníkem, kde se prezentují dosavadní výsledky a implementovaná funkcionalita. Po každé iteraci je software spustitelný, otestovaný a integrovaný. Po finálním testování, tedy v poslední iteraci, novou funkcionalitu testuje zákazník. Jedná se o tzv. akceptační testy na straně zákazníka.



Obrázek 2: Činnosti a jejich přibližná časová náročnost v jednotlivých iteracích (Eeles a Cripps, 2011)

Agilní metodiky

V posledních letech enormně vzrostl zájem o agilní metodiky. Existuje mnoho agilních metodik, například Scrum, extrémní programování apod. I když mezi jednotlivými metodikami jsou určité rozdíly, všechny vychází z podobných základních principů. Mezi tyto principy se řadí především upřednostnění funkčního softwaru před rozsáhlou dokumentací, důraz na spolupráci se zákazníkem a zejména možnost ovlivnit a inovovat procesy v rámci vývoje a dbát na názory jednotlivých členů týmu (Eeles a Cripps, 2011).

V současné době spousta firem přechází z dosavadní metodiky na agilní metodiku. Je však potřeba předem specifikovat, jakým způsobem se bude provádět. Aby agilní metodika byla efektivní, je nutné, aby se do procesu aktivně zapojil i zákazník.

Scrum

Scrum je jednou z agilních metodik. Může mít mnoho forem a modifikací. Základní popis vývoje v scrumu bude popsán v následujících řádcích.

Základní ideou scrumu je rozdělení pracovníků do menších týmů, které by měly dosahovat maximálně 10 osob. Každý tým má svého Scrum Mastera. Scrum Master se primárně stará o organizační a administrativní stránku týmu, pomáhá s překonáváním překážek, které brání v úspěšném dokončení úkolu. Není ale výjimkou, pokud Scrum Master provádí analýzu, vyvíjí nebo testuje.

Vývojová fáze se nazývá sprint. Jeho náplň je podobná jako iterace. Sprint trvá předem určenou dobu, standardní časový úsek je 14 dní. Před začátkem sprintu probíhá tzv. bodovací schůzka. Na této schůzce je přítomen analytik, který týmu prezentuje jednotlivé úkoly, které jsou z hlediska návrhu připraveny k implementaci. Jedná se o menší dílčí části, které se nazývají User Story. Ty jsou součástí tzv. Epic. Epic obsahuje několik User Story. Po prezentaci User Story tým boduje časovou náročnost, kde se zohledňují veškeré aktivity, které budou potřeba k úspěšnému dokončení úkolu, jak z vývojového, tak testovacího hlediska.

Jakmile začne sprint, začíná vývoj, příprava na následné testy a případné diskuze s analytikem. Každý den probíhá tzv. StandUp, kde každý člen týmu prohlásí, jaká byla jeho náplň práce předchozí den a jaká bude jeho plánovaná činnost na aktuální den. V momentě, kdy je dokončen vývoj User Story, přichází na řadu její otestování. Po otestování se provede akceptování funkčnosti s analytikem. Následně dochází k akceptaci se zákazníkem. Poté se provede nasazení provedených změn v rámci User Story na integrační prostředí, kde se následně provedou integrační testy.

Ke konci sprintu se pořádá týmová retrospektiva, kde se každý člen týmu může vyjádřit k průběhu sprintu a vznést případné připomínky, inovace apod.

Hlavní výhoda scrumu spočívá především ve velmi rychlé komunikaci a možnosti efektivního řešení problému. Všichni jsou členové jednoho týmu a společně zodpovídají za dokončení jednotlivých User Story. Snaha o dodržení stanovených termínů a spolupráce s ostatními členy je v tomto případě ještě více posílena.

2.3 Role v testování

Role v testování úzce souvisí s metodikou vývoje, která se na projektu používá. V některých metodikách, jako jsou agilní, tester může zastávat více rolí, včetně vývojových. Může mít na starosti nasazovaní aplikací na nové databáze pro nový sprint, automatizování testů, vyhodnocování automatických testů apod. Oproti tomu v metodikách, které vycházejí z vodopádového modelu, jsou testovací role rozděleny. Obecně lze však určit několik rolí, které se vyskytují v oblasti testování.

Tester

Tester má na starosti zpravidla provádění všech typů manuálních testů. Testuje dle scénářů, které popisují požadované chování softwaru. Pokud tester nalezne chybu,

vytvoří záznam o chybě do systému, který je určen k reportování chyb. Po opravě je chyba znovu otestována. Tester spravuje uživatelskou dokumentaci, která je používána pro dohledání požadovaného chování produktu. Slouží i jako seznámení s produktem pro nově příchozí testery. Taktéž může sloužit jako uživatelská příručka pro nově příchozí pracovníky na straně zákazníka.

Technický tester

Technický tester automatizuje funkcionalitu systému. Hlavní náplní práce je automatizace stávající funkcionality. Stará se o veškeré technické oblasti, které souvisí s testováním. Vyhodnocuje denní automatické smoke testy (testy základní a kritické funkčnosti produktu). Dále může provádět výkonnostní testování. Výkonnostní testování je zaměřeno zejména na simulaci běhu produktu pod velkou zátěží. K tomu jsou používány nástroje, které dokážou nahradit velké množství reálných uživatelů.

Test analytik

Test analytik na základě analytického návrhu funkčních požadavků od zákazníka vytváří a definuje testovací scénáře a testovací případy. Je potřeba specifikovat, které varianty se budou testovat, v jakém rozsahu se budou testy provádět. V případě nové funkčnosti zodpovídá dotazy testerů, komunikuje se zákazníkem o stavu testování. Často také funguje jako podpora pro technické testery v oblasti logiky fungování testovaného produktu.

Vedoucí týmu testování

Vedoucí týmu testování zodpovídá za otestovaný produkt a proces testování. Účastní se schůzek se zákazníkem, alokuje kapacity pro testování. Vytváří testovací plán a kontroluje jeho plnění dle stanoveného cíle. Organizuje testovací schůzky, je komunikačním prostředníkem mezi managementem společnosti a týmem. Nastavuje systém pro reportování chyb pro aktuální testovanou verzi programu.

Manažer testování

Manažer testování je zodpovědný za všechny týmy, které pod něj spadají. Může tedy být zodpovědný jak za manuální testovací oddělení, tak za technické oddělení. Přebírá odpovědnost za veškerý proces testování. Vyřizuje požadavky týmů, určuje metodiku testování, definuje sestavení testovacích týmů a jejich kapacity.

2.4 Způsoby testování

Existuje několik způsobů testování. Níže budou představeny základní způsoby testování.

Černá skříňka

Při testování černé skříňky není známa vnitřní struktura programu, programový kód apod. Tester má k dispozici testovací scénáře, které popisují jednotlivé kroky testu. Produkt se testuje s uživatelského hlediska. Není potřeba vědět, jak byl produkt implementován a není potřeba znalost porozumění a analýzy programovacího kódu.

Bílá skříňka

Bílá skříňka je opakem černé skříňky. Při testování má tester možnost nahlédnout do zdrojového kódu. Je žádané, aby tester dokázal analyzovat kód a porozuměl tomu, co která metoda způsobuje. Na základě těchto informací dokáže produkt otestovat.

Nevýhodou bílé skříňky je situace, kdy tester testuje produkt pouze podle zdrojového kódu. Výsledek takového testování nezaručuje, že budou otestovány a pokryty situace, které by mohly být objeveny při testování pomocí černé skříňky. Tento typ testování většinou nepokryje situace, které mohou nastat při reálném užívání produktu uživatelem.

Šedá skříňka

Testování pomocí šedé skříňky je kombinací bílé a černé skříňky. Může se jednat například o test v uživatelském rozhraní, kde jsou testované hodnoty porovnávány a kontrolovány s údaji v databázi. Pomocí této metody testování lze dosáhnout velmi dobrých výsledků a výrazně většího rozsahu pokrytí událostí, které mohou v produktu nastat.

Statické a dynamické testování

Statické testování nevyžaduje spuštění testovaného produktu. Do statického testování lze zařadit například analýzu kódu, požadavků apod.

Dynamické testování již vyžaduje spuštění testovaného produktu. Je potřeba spustitelná verze produktu.

2.5 Typy testů

Jednotkové testování

Jednotkové testování je proces spouštění a ověřování částí systému, aby se zjistilo, zda plní svou funkci. Testování je zaměřeno na kontrolu funkce, procedury, proměnné, třída atd. Jednotkové testování je společně s používáním nástrojů pro správu verzí zdrojových kódů nezbytné během provádění zásahů do kódu. Jednotkové testy píšou ve většině případů programátoři (Lewis, Dobbs a Veerapillai a Baker, 2009).

Integrační testování

Integračním testování se rozumí otestování integrity mezi jednotlivými moduly. V této fázi se na integrační prostředí nasazují části, které již byly otestovány samostatně. Testuje se především chování a komunikace mezi přidanými částmi. Integrační testování je důležitý prvek. Jednotlivé komponenty mohou samostatně fungovat v pořádku, ale pokud je přidán další modul, můžou nastat komplikace.

Do fáze integračních testů spadají tzv. Smoke testy. Smoke test je zaměřen na hlavní funkcionalitu systému. Testují se pouze ty části, bez kterých by byl produkt v kritickém, nepoužitelném stavu. U smoke testů je velmi vhodné implementovat jejich automatizaci. Poté lze smoke testy automaticky pouštět například v noci, kdy se provádí průběžné nasazování nové funkcionality.

Systémové testování

Systémové testování je hloubkový test systému, při kterém se testují veškeré možné aspekty systému. Systémové testy mohou pokrývat následující oblasti, které jsou popsány v (Kenett a Baker, 1999) a známy pod zkratkou FURPS:

- **Functionality** (Funkčnost) – produkt má všechny požadované funkce, které byly požadovány zákazníkem včetně zachování stávající funkcionality.
- **Usability** (Použitelnost) – produkt lze používat bez výrazných obtíží, je intuitivní a je dodána patřičná dokumentace a případné školící materiály.
- **Reliability** (Spolehlivost) – vztahuje se na frekvenci a závažnost chyb, jak se může uživatel spolehnout na výsledky programu apod.
- **Performance** (Výkon) - zaměřuje se na rychlost, s kterou program vykonává požadované operace, simuluje se přístup většího množství uživatelů apod.
- **Serviceability** (Udržovatelnost) – sleduje se hladký průběh instalačního procesu, přizpůsobitelnost a možnosti budoucího rozšíření.

V systémovém testování se produkt testuje jako funkční celek. Testuje se jak stávající, tak nová funkcionalita. Testy stávající funkcionality se nazývají regresní testy. Regresní testy zahrnují veškerou dosavadní funkcionalitu. Provádějí se z toho důvodu, aby se ověřilo, že nově přidaná funkcionalita neovlivnila funkcionalitu stávající. Regresní testy jsou velmi rozsáhlé a jelikož se jedná o opakované provádění totožných scénářů, je vhodné je zautomatizovat.

Akceptační testování

Akceptační testování probíhá na straně zákazníka. Tyto testy většinou provádí zákazník. Rozsahem jsou na podobné úrovni jako regresní testy. Akceptační testy jsou testovány především na situacích, které reálně nastávají. Může se tedy stát, že jsou

objeveny chyby, které nebyly odhaleny v regresních testech. Pokud tato situace nastává často, je vhodné tzv. předání znalosti, kde zákazník prezentuje, jak se používá produkt na produkci, kde jsou kritické části apod.

2.6 Testovací dokumentace

Testovací plán

Testovací plán je forma dokumentu, která určuje a popisuje veškeré informace o plánovaném testování a událostech, které mohou nastat. Testovací plán může mít mnoho podob. Jednotlivé body a obsah testovacího plánu dle (IEEE, 2008) je definován následovně:

- **Identifikace testovacího plánu** – jedná se o jednoznačnou identifikaci testovacího plánu. Je potřeba použít unikátní identifikátor, pomocí kterého lze určit, jaký testovací plán je přidružen k určitému projektu.
- **Reference** – v této části se odkazuje na dokumenty, které mají určitou souvislost s testovacím plánem. Může to být například projektový plán, vývojové a testovací standardy apod.
- **Úvod** – v sekci úvod je vysvětlen a přiblížen hlavní účel testovacího plánu. Je vhodné uvést základní informace o produktu, který bude testován. Může se zde nacházet i informace o rozpočtu na projekt.
- **Seznam testovaných položek** – definuje a jednoznačně identifikuje seznam produktů a jejich verze, které budou v rámci plánu testovány.
- **Vlastnosti, které budou testovány** – obsahuje seznam požadovaných vlastností, které budou testovány z uživatelského hlediska. Nejedná se o technický popis softwaru, ale o uživatelský pohled testované funkcionality.
- **Vlastnosti, které nebudou testovány** – obsahuje seznam vlastností, které nebudou testovány z uživatelského hlediska. Stejně jako u předešlé sekce se nejedná o technický popis softwaru, ale o uživatelský pohled testované funkcionality. Je zde uvedeno, proč a z jakého důvodu tyto vlastnosti nebudou testovány. Může to být například z důvodu malého dopadu při případném chybném chování, jedná se o stávající chování, které však nebrání provozu apod.
- **Strategie** – jedná se o komplexní přehled testovací strategie. Ve strategii je uvedeno, jestli budou potřeba speciální nástroje, jaký bude použit hardware a software. Dále je určeno, na jakém prostředí bude testování probíhat, jaký způsob testování bude zvolen pro jednotlivé typy testů. Může být specifikována i frekvence setkání a porad.

- **Kritéria pro dokončení** – v této části jsou definována a nastavena kritéria potřebná pro dokončení stanoveného plánu. Může se jednat například o stanovenou procentuální hranici úspěšně dokončených testů.
- **Přerušení testů** – zde jsou uvedeny podmínky, na základě kterých lze přerušit testování a kdy je možné testování obnovit. Mezi takové lze zařadit závažné chyby, které blokují významnou část testovacích případů a testování takového nestabilního systému nemá význam.
- **Výstupy z testování** – v části výstupy z testování je stanoveno, co má být dodáno v rámci testovacího plánu. Jedná se především o testovací plán jako celek, testovací případy, použité data při testování, zprávu o testování, použité nástroje, zprávy o komplikacích a jejich řešení apod.
- **Zbývající úkoly** – v případě, pokud se na vývoji podílí více společností a požadovaná funkcionalita není vyvinuta, je zde uvedeno, které případy nebyly otestovány.
- **Požadavky na zdroje** – v této sekci jsou uvedeny všechny speciální požadavky, například speciální hardware jako simulátor, potřebné softwarové vybavení,potřebná testovací data, požadavky na omezení produktu během jeho testování apod.
- **Školení** – v této části je uvedeno, jestli bude potřeba provést školení na testovaný produkt nebo jestli bude nutné seznámit se s určitými testovacími nástroji.
- **Odpovědnosti** – tato část identifikuje odpovědné osoby nebo týmy, které jsou odpovědné za splnění jednotlivých částí. Může jít o osoby, které rozhodují o důležitých požadavcích, provádějí školení, určují obsah testování apod.
- **Harmonogram testování** – určuje časové rozvržení testování a dosáhnutí požadovaných milníků. Na základě harmonogramu testování jsou prováděny jednotlivé fáze typů testů.
- **Plánování rizik** – Určuje, jaká mohou nastat rizika spojená s testováním. Na rozdíl od sekce rizika testovaného softwaru se zde zvažují rizika, které nejsou spojena s testovaným softwarem. Mezi taková rizika patří především nedostatek personálních zdrojů při začátku testování, pozdní dodání potřebného hardwaru, speciálních nástrojů a dat. Mezi další lze zařadit nepřipravené nebo nestabilní prostředí pro testování, zpoždění při školení, požadavky na změnu oproti původnímu návrhu apod.
- **Souhlas s testováním** – zahrnuje jména osob, která jsou odpovědná za testování a svým podpisem stvrzují testovací plán.
- **Rejstřík** – obsahuje výrazy a zkratky použité v testovacím plánu a obraty používané v testování obecně, aby se předešlo případnému nedorozumění.

Testovací případ

Testovací případ popisuje konkrétní akce prováděné s určitou softwarovou komponentou a jejich očekávané výsledky. Mohou být reprezentovány seznamem prováděných kroků a očekávaných výsledků. Testovací případ je formální dokument nebo záznam popisující, jak provést určitou testovací činnost. Testovací případy mohou ověřovat, jestli program pracuje správně dle určených podmínek. Často se zaměřují na ošetření chybových podmínek. Pokud je testovací případ dobře zdokumentovaný, může jej provést nejen tester (Page, Johnston a Rollison, 2008).

Vlastnosti, které by měl splňovat kvalitní testovací případ dle (Page, Johnston a Rollison, 2008)

- **Účel** – je potřeba vysvětlit, proč je testovací případ důležitý a k čemu slouží. Účelem může být ověření konkrétní funkcionality, kontrola ošetření chybových stavů, ověření chování v určité situaci apod.
- **Podmínky** – je potřeba popsat a definovat, jaké vlivy prostředí jsou pro provedení testovacího případu podstatné a které jsou naopak nepodstatné. Pokud je potřeba produkt spouštět například na specifickém hardwarovém vybavení, případně je k vykonaní potřebné použít další software, je nutné tyto skutečnosti do testovacího případu zakomponovat.
- **Konkrétní vstupy a kroky** – je žádoucí, aby měl testovací případ jasně definované všechny kroky, které vedou k přesnému a opakovatelnému provedení testu.
- **Očekávané výsledky** – je nutné do detailu uvést všechny informace, které jsou nezbytné k ověření, zda byl test úspěšný nebo neúspěšný.

Testovacích případů může být nepřeberné množství. Je tedy potřebné tyto případy spravovat, k čemuž se využívají nástroje určené pro správu testovacích případů. Není ovšem výjimkou, že se ke správě používá Excel. Takové řešení může být dostatečné pro menší projekty. Pokud je ale testovacích případů více a zároveň se Excel využívá pro sledování průběhu testování, můžou nastávat určité komplikace. Excel, kde jsou uložené odkazy na testovací případy, stav otestování testovacích případů atd. se může při přístupu většího množství uživatelů zamknout. Může nastat i situace, kdy se katalog v Excelu poškodí a nelze otevřít. Taktéž práce s katalogem je zdlouhavá. Než se uloží provedené změny, může taková operace trvat i několik minut. V takové situaci je rozumnější využít některý z nástrojů pro to určených. Mezi nejznámější patří například TestLink, HP Quality Center a další.

Testovací scénář

Testovací scénář obsahuje několik testovacích případů. Testovací scénář reprezentuje větší logické celky, které vzniknou sloučením nebo kombinací požadovaných testovacích případů.

Testovací report

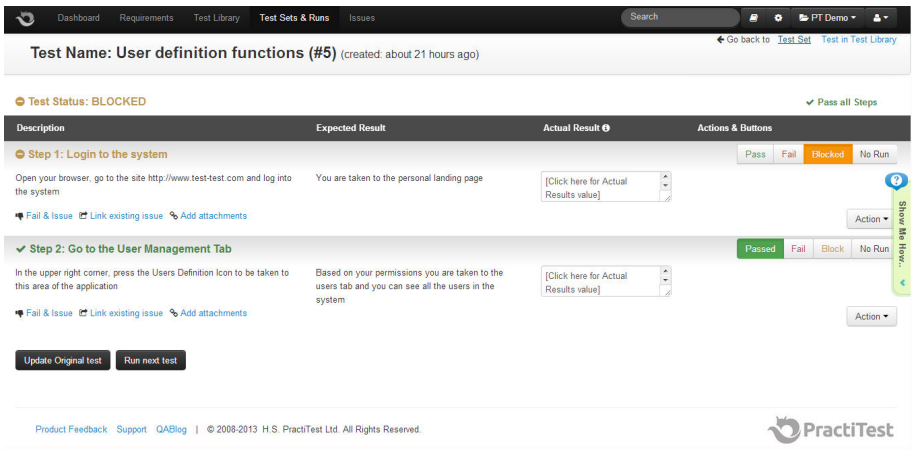
Testovací report může mít mnoho podob. Může být zpracováván denně, týdně, měsíčně apod. Zobrazuje především aktuální stav testování a odhad, kolik procent je potřeba mít otestováno k určitému datu. Obsahuje veškeré informace o dění během testování, například události, které vedly k případné prodlevě při testování, stav automatizovaných testů apod. Informuje o chybách, které mají největší prioritu a o chybách, které nejvíce blokují provedení ostatních testů.

2.7 Nástroje pro správu testovacích případů

Při vývoji nové funkcionality roste také počet nových testovacích případů. Pro jejich správu jsou určeny nástroje pro správu testovacích případů. Takový nástroj umožňuje například testovací případy vytvářet, určovat vazby mezi jednotlivými případy, sledovat provedené změny. Na základě provedených změn lze nastavit zasílání těchto informací odpovědným osobám. V následujících řádcích budou představeni někteří zástupci těchto nástrojů.

PractiTest

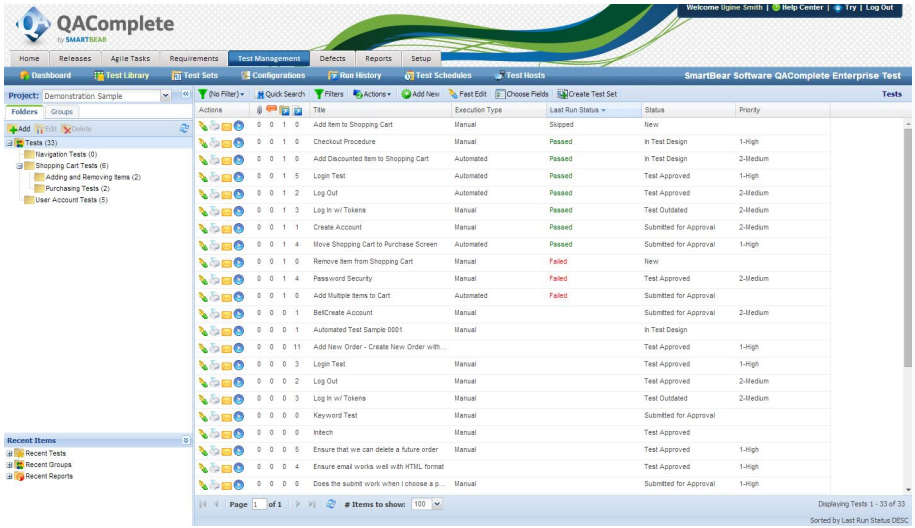
PractiTest se řadí mezi komerční nástroje. Podporuje vytváření, správu, editaci, spouštění a vyhodnocování výsledků testů. Ke každému kroku v testovacím případu lze zadat stav otestování a v případě již existující nebo blokující chyby přidat vazbu na tuto chybu. Chybu a její popis lze automaticky vyplnit na základě daného testovacího případu. Dále lze ke každému kroku testovacího případu přiložit přílohu. Zajímavou funkcí je importování a exportování testovacích případů. V nástroji PractiTest lze testovací případy řadit do hierarchické struktury. PractiTest lze integrovat s nástroji pro reportování chyb (JIRA, Bugzilla a Redmine). Pro možnost zhlédnutí živě prováděné demonstrace práce s nástrojem PractiTest a jeho funkcionalit se lze přihlásit na internetové adrese
`http://subscribe.practitest.com/users/new` a vybrat některý z nabízených termínů.



Obrázek 3: Ukázka testovacího případu v nástroji PractiTest (PractiTest, 2015)

QAComplete

QAComplete je komerční nástroj od společnosti SmartBear Software. QAComplete je plnohodnotný nástroj pro správu testovacích případů. Pomocí QAComplete lze provádět veškeré činnosti, které jsou potřebné při procesu testování. Lze vytvořit testovací plán, naplánovat časový harmonogram jednotlivých testů nebo celé skupiny, vytvářet dokumentaci a reporty. Umožňuje spravovat jak manuální, tak automatické testy (podpora Selenium testů), případně vytvářet testy, které obsahují manuální i automatické kroky. Pokud je při vykonávání jednotlivých testovacích případů objeveno neočekávané chování, lze přímo vytvořit chybu, do které se automaticky doplní informace a jednotlivé kroky vedoucí k nasimulování chyby. QAComplete lze integrovat s nástroji pro reportování chyb (JIRA). QAComplete lze vyzkoušet zdarma po dobu 30 dní.



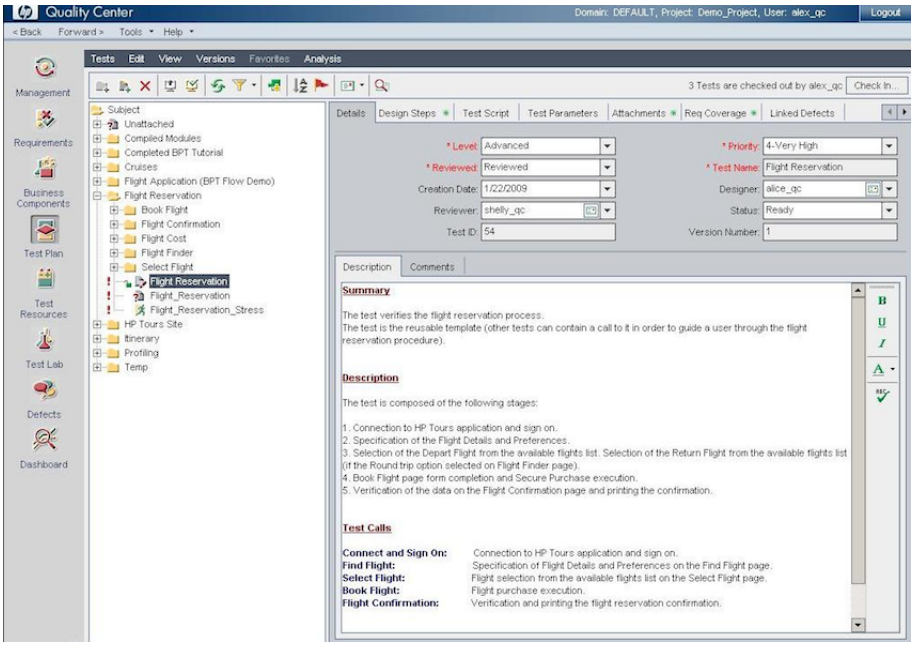
Obrázek 4: Ukázka správy testovacích případů v QAComplete (SmartBear, 2015)

Quality Center

Quality Center je komplexní komerční sada nástrojů od společnosti Hewlett-Packard. Mezi nabízenou funkčnost patří i správa testovacích případů a plánů. Podporuje plánování testů a analýzu výsledků. K testovacímu případu lze uložit a zaznamenat například:

- popis testovacího případu včetně detailních údajů,
- postup vedoucí k žádoucímu výsledku,
- testovací skript,
- seznam potřebných testovacích parametrů,
- případné přílohy,
- informaci o tom, jestli je na testovací případ aktuálně založena chyba.

Quality Center obsahuje i podpůrné nástroje pro automatizaci testů, správu chyb a reportování.



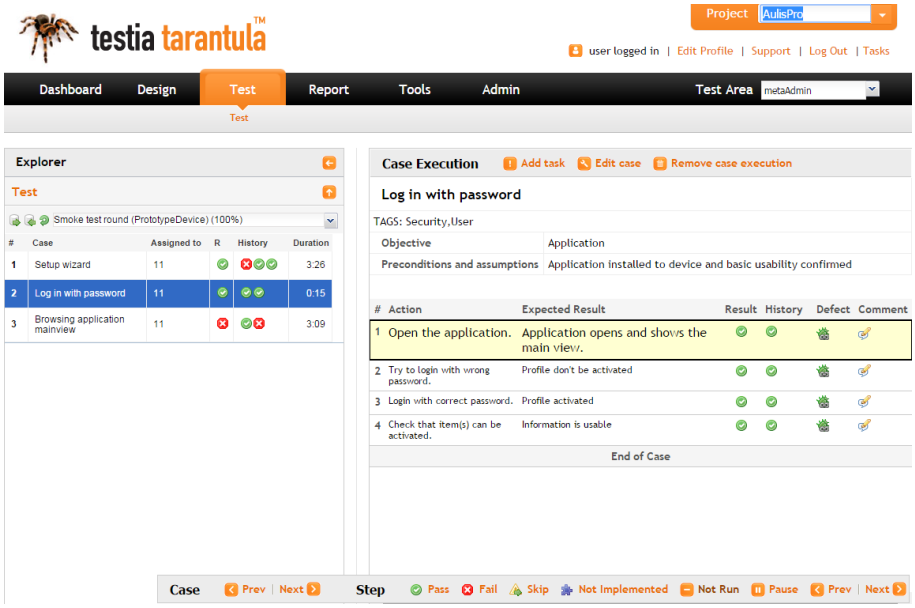
Obrázek 5: Ukázka testovacího případu v nástroji Quality Center (Testhouse, 2014)

Tarantula

Tarantula je bezplatně poskytovaný nástroj pro správu testovacích případů v agilním vývoji. Tarantula je přístupná přes webové rozhraní. Dokáže vytvářet, spravovat a spouštět testovací případy včetně následného reportování stavu testování. Tarantula může být integrována s nástroji pro reportování chyb (JIRA a Bugzilla).

Nástroj lze vyzkoušet na internetové adrese `http://pts.tarantula.fi/t/home/login`, přihlašovací údaje jsou `user/user`. Po přihlášení jsou k dispozici následující záložky:

- **Dashboard** – zobrazení celkového přehledu, denního postupu, otevřených chyb, výsledky testů apod.
- **Design** – vytváření a správa testovacích případů.
- **Test** – testování jednotlivých testovacích případů.
- **Report** – možnost tvorby reportů o výsledku testování.
- **Tools** – export a import testovacích případů.
- **Admin** – nastavení a správa projektů, uživatelů apod.

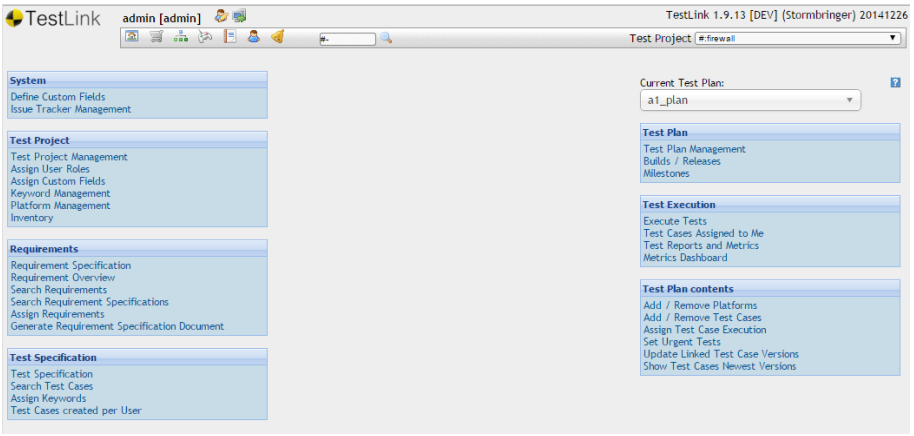


Obrázek 6: Ukázka práce s testovacími případy

TestLink

TestLink se řadí mezi bezplatně poskytované nástroje pro správu testovacích případů. TestLink je přístupný přes webové rozhraní. Testovací případy lze členit do hierarchické struktury. Uživatelé mohou k jednotlivým položkám přiřazovat klíčová slova a snadno tak filtrovat požadované položky. Jednotlivé testy mohou být přiřazeny určeným uživatelům. Samozřejmostí je podpora generování reportů o průběhu testování. Testovací plány, případy apod. lze importovat a exportovat. TestLink lze integrovat s nástroji pro reportování chyb (JIRA, Bugzilla a Redmine). TestLink komunita vytvořila

a zpřístupnila přehledný uživatelský manuál dostupný na internetové adrese `https://wiki.openoffice.org/w/images/1/1b/Testlink_user_manual.pdf`. TestLink lze vyzkoušet na internetové adrese `http://demo.testlink.org`, přihlašovací údaje jsou `admin/admin`.



Obrázek 7: Ukázka hlavní stránky a dostupných možností nástroje TestLink

2.8 Nástroje pro reportování chyb

Pro hlášení a zaznamenávání chyb je potřebné použít patřičný nástroj. Lze se obejít i bez něj, ale velmi to komplikuje proces opravy chyby, obzvlášť pokud se jedná o větší projekt, do kterého je zapojeno více osob. V takovém případě může chyba, která byla nahlášena například pouze pomocí emailové komunikace, lehce zapadnout. Nástroj pro reportování chyb by měl umožňovat vytvářet jednotlivé projekty pro zadávání chyb. U zadané chyby by mělo být podporováno přiřazení na určené osoby, změna stavu, přidání komentářů, sledování historie změn a možnost sledovat chybu. To znamená, že v případě jakýchkoli změn je odeslána notifikace, například na email. Při vytváření defektu jsou důležité následující údaje, které ulehčí identifikaci příčiny chyby:

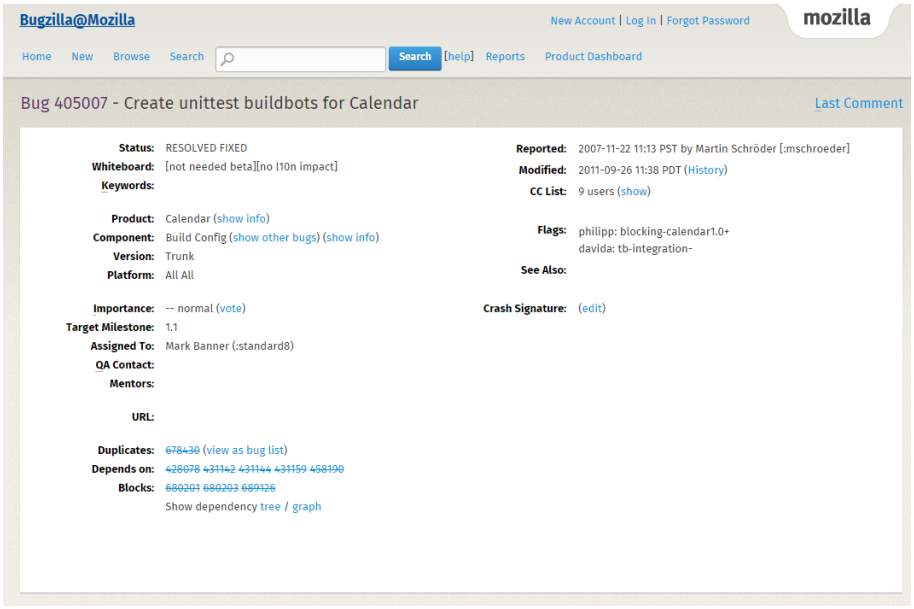
- stručný název chyby,
- aktuální testovací cyklus,
- prostředí, na kterém se chyba vyskytla,
- verze testovaného produktu,
- detailní popis kroků vedoucí k chybě,
- priorita a závažnost,
- případná příloha (fotka obrazovky v době chyby, textová chybová hláška apod.).

V následujících řádcích budou přiblíženy jednotlivé nástroje pro reportování chyb.

BugZilla

BugZilla je bezplatně poskytovaný nástroj pro reportování chyb. Umožňuje přístup přes webové rozhraní. Dovoluje nastavit vazby mezi jednotlivými chybami. Lze nastavit odkaz na duplicitní, blokující nebo návaznou chybu. Při vytváření lze nahrát přílohy. Nechybí možnost přidávání komentářů a sledování chyby. Každá chyba má své unikátní číselné označení. Výsledné URL pro chybu s označením 123 je ve tvaru `.../show_bug.cgi?id=123`, v případě jiného defektu se mění pouze číselný identifikátor chyby, což umožňuje univerzálnost při odkazování na jednotlivé chyby.

Testovací verze je umístěna na internetové stránce `https://landfill.bugzilla.org`. Na testovací verzi lze libovolně zadávat chyby, vytvářet reporty a zkoušet veškerou dostupnou funkcionalitu.

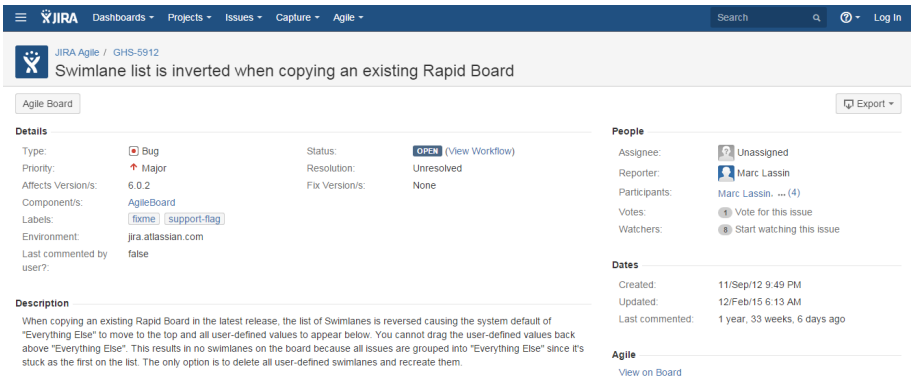


Obrázek 8: Ukázka zadané chyby v nástroji BugZilla

JIRA

JIRA je komerční nástroj od společnosti Atlassian. Jedná se o nástroj, díky kterému je možné nejen zadávat chyby, ale také kompletně plánovat a řídit projekty. JIRA je nástroj, který lze přizpůsobit dle potřeb. Lze vytvářet jednotlivé projekty. Na úvodní straně projektu se mohou nacházet informace o stavu testování, otevřených chybách apod. Na jednotlivé defekty, úkoly pro vytvoření nové funkcionality apod. lze vykazovat odpracovaný čas. Na základě těchto údajů lze poté sledovat odvedenou práci.

JIRA disponuje propracovaným vyhledáváním, například pomocí následujícího příkazu se vyhledají vytvořené defekty, seřazené od nejnovějšího po nejstarší, které vytvořil aktuální uživatel – `reporter = currentUser() ORDER BY createdDate DESC`. Testovací verzi lze vyzkoušet na internetové stránce `https://jira.atlassian.com`.

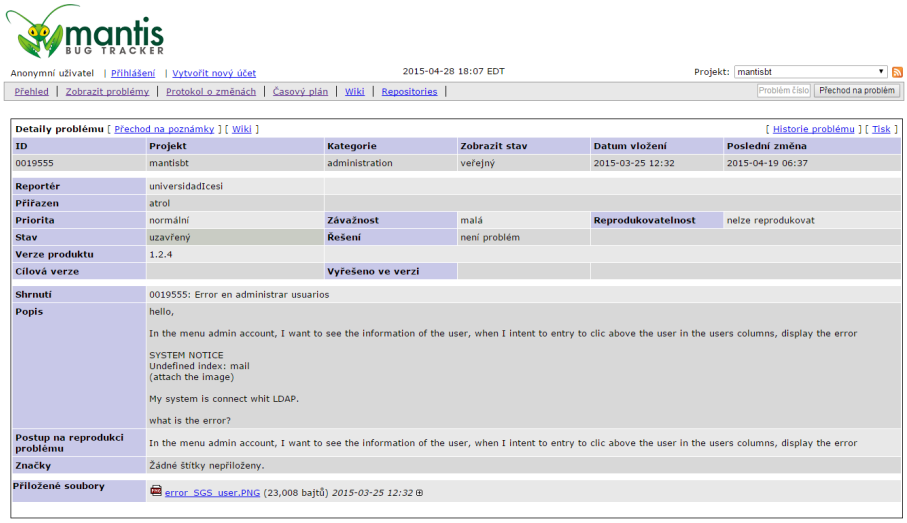


Obrázek 9: Ukázka zadané chyby v nástroji JIRA

MantisBT

MantisBT je bezplatně poskytovaný nástroj. Umožňuje vytvářet jednotlivé testovací projekty. Na úvodní stránce jsou zobrazeny informace o aktuálních chybách, nedávných aktivitách u jednotlivých defektů apod. Z hlavní stránky lze po zadání čísla požadované chyby přejít přímo na požadovaný defekt. Stejně jako u předešlého nástroje je URL adresa v případě čísla chyby 123 generována ve tvaru `.../view.php?id=123`. Mezi další funkce lze zařadit tvorbu reportů, grafů, export zadaných chyb apod.

MantisBT lze vyzkoušet na internetové stránce `https://www.mantisbt.org/bugs`.



Obrázek 10: Ukázka zadané chyby v nástroji MantisBT

3 Automatizované testování

Automatizované testování nemůže v žádném případě v plném měřítku nahradit testování manuální, pouze pomáhá odvádět práci testerů jednodušeji, rychleji a efektivněji (Patton, 2006).

Automatizované testování má smysl v případě, pokud budou požadované testovací případy spouštěny ve větším měřítku. Před samotnou implementací automatizovaných testů je vhodné mít k dispozici již existující manuální testovací případ, podle kterého se vytvoří samotný automatizovaný test. Automatizované testy je vhodné vytvářet například pro regresní testy (testy stávající funkcionality) a smoke testy (testy základní a kritické funkčnosti produktu). V případě automatizace testů nové funkčnosti je zapotřebí vzít v úvahu, jestli se do plánované oblasti neplánuje přidávat další prvky, upravovat a vylepšovat stávající chování apod. Stejně tak je zapotřebí zvážit, jak náročné je provést automatizaci na požadovaném produktu. Například implementace automatizovaných testů pro webové aplikace je ve většině případů méně náročná než u desktopových aplikací.

3.1 Výhody a nevýhody automatizovaného testování

Výhody automatizovaného testování

Jedna z největších výhod při automatizovaném testování je rychlost, s jakou je testování prováděno. Lze otestovat velké množství nejrůznějších průchodů a kombinací, které by v případě ručního testování zabralo nezanedbatelnou část testovacích kapacit. V některých případech by takový test nebylo možné provést ručně, pokud by se jednalo například o testování v řádech několika tísíců odlišných kombinací. Automatizovaný test tento proces zvládne v závislosti na složitosti testovaného systému podstatně rychleji a může je provádět v každém testovacím cyklu opakovaně.

Další výhodou oproti manuálnímu testování je přesnost a identické provedení každého jednotlivého automatizovaného testu i v případě jeho opětovného spuštění. Automatizovaný test vykonává testovací případy vždy stejně a simuluje vždy totožné množství předem nadefinovaných průchodů v testovaném systému. V případě manuálního testování není zaručeno provedení každého testovacího případu ve stejném rozsahu. Při velkém množství testovacích případů se může dostavit únava a nepozornost, která může vést k přehlédnutí neodpovídající hodnoty, neprovedení všech možných kombinací apod.

Nevýhody automatizovaného testování

Mezi nevýhody automatizovaných testů lze zařadit jejich náročnost na případnou údržbu, pokud je do testovaného produktu přidána nová funkčnost nebo je přepsána stávající funkčnost. Může se vyskytnout i situace, kdy je některý modul produktu přepsán do jiného programovacího jazyku.

Automatizovanými testy prozatím nelze plnohodnotně ve všech případech nahradit manuální testování prováděné testerem. Automatizovaný test bude provádět a kontrolovat pouze ty věci, pro které je naprogramován a pro které má jasně definované instrukce. Na rozdíl od manuálního testování, kde může tester vyzkoušet kombinace, které nejsou definované v analýze funkčnosti programu a tím pádem lze objevit chybné chování. Pokud se při testovacím případu, který je prováděn pomocí automatizovaného testu, objeví chybné chování, pro které není automatizovaný test přizpůsoben, nemusí se toto chybné chováni rozpoznat a test se vyhodnotí jako správný. Pokud by se spoléhalo ve všech případech testování pouze na automatizované testy, chyba by se neobjevila v rámci testů v testovacím prostředí, ale například až v rámci akceptačního testování, které provádí zákazník. V horším případě by se na chybné chování mohlo přijít až na produkčním prostředí, kde je proces opravy chyby náročnější jak z časového, tak tím pádem i finančního hlediska.

3.2 Techniky v procesu automatizovaného testování

V následujících řádcích budou představeny některé techniky, které lze využít při vytváření a správě automatizovaných testů.

Záznam aktivity uživatele

Tato technika se řadí mezi nejznámější formy automatizace. Za určitých okolností umožňuje dosáhnout uspokojivých výsledků i bez hlubších technických znalostí. Princip je velmi jednoduchý. Testovací nástroj zachycuje veškeré aktivity uživatele provádějícího jednotlivé testovací kroky. Po ukončení záznamu umožňuje jejich opětovné přehrání. Zaznamenané testovací kroky jsou však v případě provedení jakékoli změny v aplikaci velmi náročné na údržbu. Využití je vhodné pouze v případě, pokud se do testované oblasti neplánuje zasahovat, případně jen v minimální míře (Roudenský a Havlíčková, 2013).

Úprava vygenerovaných skriptů

V tomto případě se pro vytvoření testu vychází z vygenerovaného skriptu ze záznamu aktivity uživatele. Ten je poté možné upravit podle požadovaných potřeb. Lze implementovat složitější logiku a provést tak rozšíření obsahu testu a celkově dosáhnout lepší udržovatelnosti a znovupoužitelnosti daného skriptu (Roudenský a Havlíčková, 2013).

Testování řízené daty

Testování řízené daty je vhodné v situacích, kdy je potřeba provést opakovaně testy, které jsou založené na stejném principu, ale obsahují velké množství různých vstupů a výstupů. Data mohou být generována pomocí metod přímo v implementovaném

kódu nebo z externího skriptu, načítána ze souboru, získávána pomocí dotazů z databáze apod. (Roudenský a Havlíčková, 2013).

Testy řízené klíčovými slovy

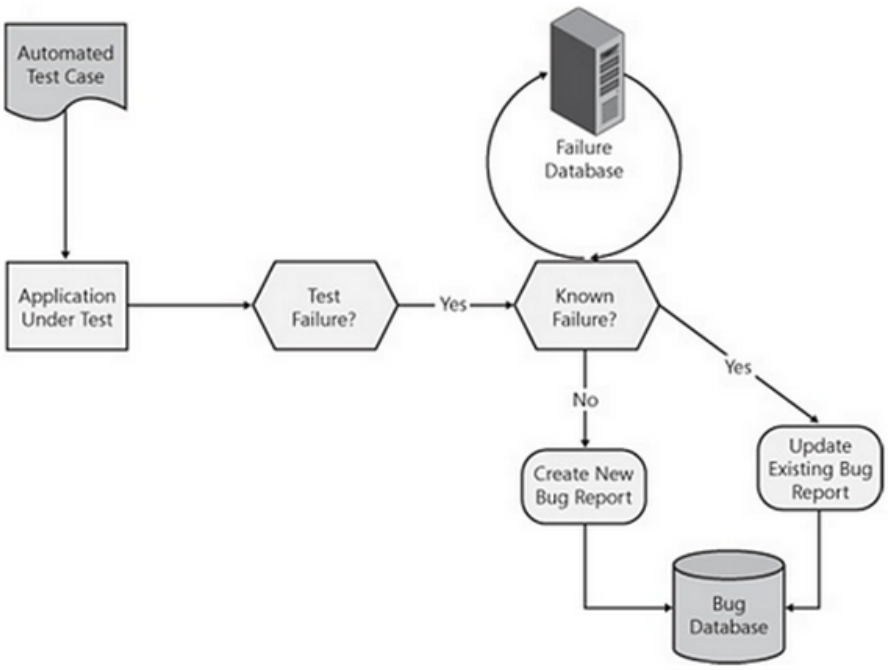
Testy řízené klíčovými slovy jsou tvořeny pomocí vstupních dat a příkazů, ze kterých je tvořen testovací skript - klíčová slova. Klíčová slova představují obecné akce v uživatelském rozhraní, může jít například o přihlášení do systému, zadání hodnoty apod. Z klíčových slov lze poté jednoduše poskládat testovací případ. Jejich požadované chování je nutné předem implementovat. Poté lze sestavovat jednotlivé testovací případy pomocí klíčových slov (Roudenský a Havlíčková, 2013).

Kombinovaný přístup

Kombinovaný přístup technik zmíněných výše je využíván nejčastěji. Pro určitou situaci nemusí některá z technik stačit a je potřeba ji doplnit o další, aby bylo dosaženo požadovaného výsledku. Jde tedy především o využití předností jednotlivých technik (Roudenský a Havlíčková, 2013).

Automatická analýza selhání automatizovaných testů

Při opakovaném spouštění většího množství testovacích případů na různých prostředích v krátkém časovém intervalu může nastat situace, kterou lze popsat jako analytická paralýza. Při vyhodnocení automatizovaných testů, které selhaly, může toto vyhodnocení a případné opětovné spuštění zabrat nepřeberné množství času. Řešením této situace může být automatická analýza selhání testů. Pokud dojde k selhání testovacího případu na testované aplikaci, dojde k porovnání v databázi selhání testů. K tomu je zapotřebí, aby automatizovaný test dokázal poskytovat údaje o testovacích krocích, které vedly k selhání, použitém testovacím prostředí apod. V případě nalezení známého selhání je existující záznam aktualizován v databázi chyb. Pokud se jedná o nový typ selhání testu, je vytvořen nový záznam o chybě. Pomocí této metody lze zefektivnit proces vyhodnocení chyb a vykonávat tak práci kvalitněji. Na obrázku 11 je demonstrován návrh takového systému (Page, Johnston a Rollison, 2008).



Obrázek 11: Návrh automatické analýzy selhání testů (Page, Johnston a Rollison, 2008)

SEARCH

SEARCH je označení pro techniku používanou při práci s automatizovanými testy. Popisuje jednotlivé kroky a součásti, které může obsahovat automatizovaný test. Jednotlivé kroky dle (Page, Johnston a Rollison, 2008) jsou následovné:

- **Setup** (nastavení) – nastavení slouží pro uvedení testovaného produktu a testovacího prostředí do takového stavu, kdy je možné provést požadované kroky testu.
- **Execution** (provedení) – provedení jednotlivých kroků automatizovaného testu vedoucí k ověření funkcionality testovaného produktu.
- **Analysis** (analýza) – analýza slouží zejména pro zjištění, jestli byl test úspěšný nebo selhal.
- **Reporting** (reportování) – obsahuje zobrazení a publikování výsledků analýzy, například vytvořením souborů, odeslání emailů apod.
- **Cleanup** (úklid) – uvedení testovaného produktu nebo testovacího prostředí do stavu, který bude umožňovat provedení dalších testů.
- **Help** (dokumentace) – dokumentace popisuje účel testu, omezení, poznámky o potřebné konfiguraci, návod k interpretaci požadovaných výsledků apod.

3.3 Nástroje pro automatizované testování

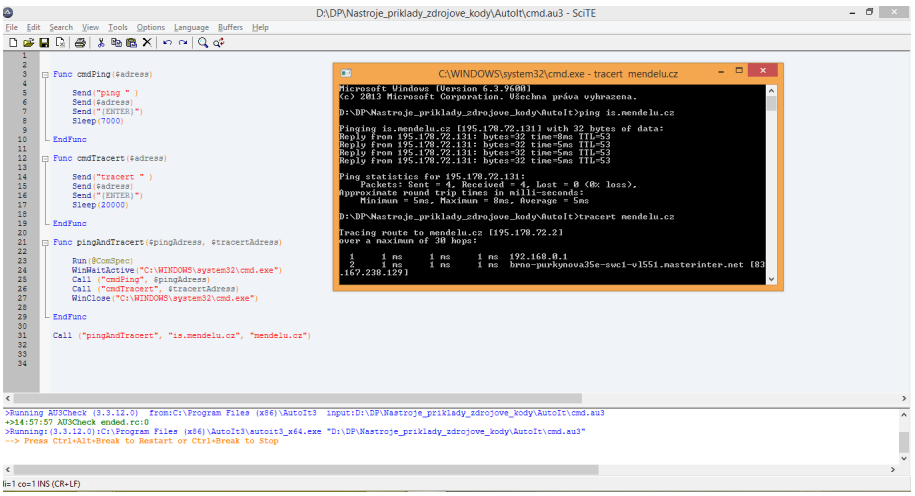
V následujících řádcích budou představeny nástroje pro automatizované testování a budou vyzkoušeny jejich základní funkce. V rámci jednotlivých nástrojů budou vytvořeny příklady, které demonstrují možnosti zkoumaných nástrojů.

AutoIt

AutoIt je nástroj, který je poskytován zdarma. Je určen pro automatizaci opakujících se činností. Podporuje simulaci stisků jednotlivých klávesnic, kliknutí myši apod. Taktéž dovoluje manipulaci s okny, například změnu jejich velikosti. Dokáže spouštět aplikace ve Windows pomocí příkazu `Run("nazevProgramu.exe")`. Metoda `WinWaitActive("nazevOknaProgramu")` poté počká, dokud není okno spuštěného programu aktivní. Poté již lze například pomocí `Send("text")` simulovat psaní textu. Metodou `WinClose("nazevOknaProgramu")` lze zavřít program.

Jedná se o velmi jednoduchý nástroj, který ke svému běhu nepotřebuje dodatečné externí `.dll` knihovny a nezasahuje do registrů. Napsané testy mohou být zkompilovány jako samostatně spustitelné `.exe` soubory pomocí Aut2Exe, který je instalován společně s AutoIt. Zkompilované skripty lze poté spustit i tam, kde není AutoIt nainstalován. Testy jsou psány pomocí skriptovacího jazyka, který je založen na jazyku BASIC. Přehledně zpracovaná uživatelská dokumentace je dostupná na internetové adrese `https://www.autoitscript.com/autoit3/docs`.

Na obr. 12 je ukázka jednoduchého skriptu, který otevře příkazovou řádku a provede příkazy `ping` a `tracert` na definované adrese. Příkazová řádka se po provedení zavře.



Obrázek 12: Ukázka testovacího skriptu v prostředí AutoIt

HP Unified Functional Testing

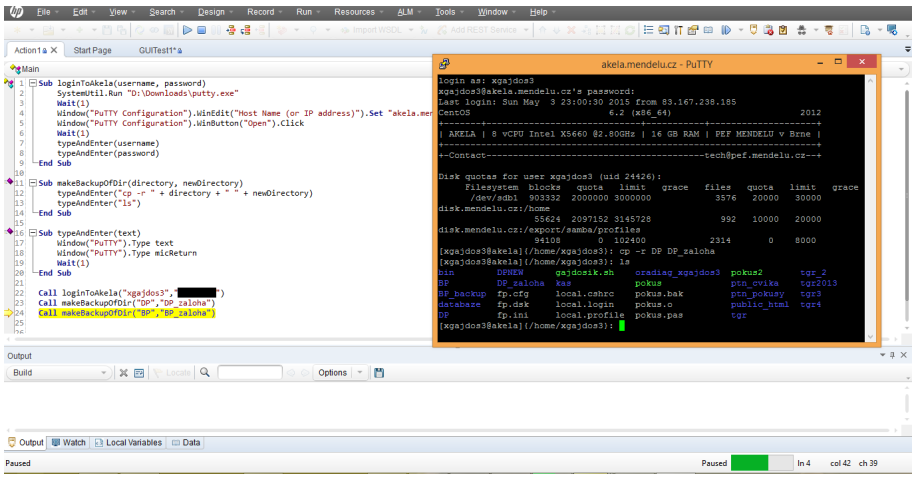
HP Unified Functional Testing je komerční nástroj od společnosti Hewlett-Packard. Představuje komplexní řešení pro automatizaci funkčních testů. HP UFT podporuje psaní testů v jazyce VBScript a lze je vytvářet pro webové aplikace, Windows aplikace a mobilní aplikace. Testy lze vytvářet i za pomocí nahrávání akcí, které provádí uživatel. Následně je možné nahrané akce upravit dle potřeby v editačním prostředí. Při tvorbě testů je možné využít velké množství nabízené funkcionality.

Testovací případ v HP UFT představuje akci. Akce lze mezi sebou provázat, nastavit jim požadované parametry, např. počet opakování apod. Akce a jejich návaznosti mezi sebou lze vizuálně reprezentovat ve formě vývojových diagramů. HP UFT dovoluje použít pro testy data, které lze importovat například ze souboru z aplikace Excel.

Po spuštění testu se na základě právě vykonané akce interaktivně podbarvuje řádek kódu, který momentálně vykonává akci. Testy lze také pohodlně odlaďovat a nastavovat kontrolní body. Po dokončení testu je vygenerován testovací report, který obsahuje přehledné informace o výsledku testu včetně vygenerovaných grafů.

Pro HP UFT je k dispozici velmi rozsáhlá uživatelská dokumentace na internetové adrese `http://goo.gl/dNcdAV`.

Na obr. 13 lze vidět prostředí HP UFT. Byl zde vytvořen jednoduchý test, pomocí kterého je provedeno přihlášení na server `akela.mendelu.cz` pod definovaným uživatelským jménem pomocí programu PuTTy . Následně je vytvořena kopie zadaných adresářů a jsou vypsány všechny adresáře a soubory.



Obrázek 13: Ukázka testu v HP Unified Functional Testing

RiaTest

RiaTest je komerční nástroj od společnosti Cogitek. RiaTest podporuje tvorbu testů pro následující technologie:

- Adobe Flex aplikace,
- Windows aplikace,
- Windows 8 Store aplikace,
- webové aplikace (Javascript, HTML5, jQuery, ExtJS).

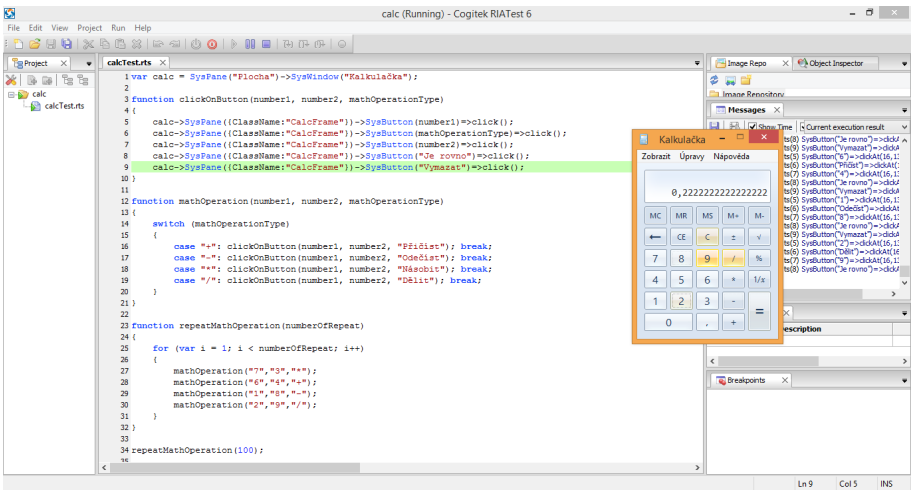
Testy lze realizovat za pomocí skriptovacího jazyka Riascript, který je podobný jazyku Javascript. Při implementaci testů probíhá na pozadí kontrola syntaxe.

Implementace testů je možná za pomocí záznamu akce uživatele, který je poté převeden do podoby skriptu, který lze dále upravovat. Skripty lze ladit a jednodušeji tak odhalit případné chyby pomocí kontrolních bodů a krokování. V rámci RiaScript lze ošetřit a zachytit chybový stav pomocí výjimek. Spuštěný test lze pozastavit, upravit jeho implementaci a znovu jej spustit bez nutnosti spouštět celý test znovu. RiaTest dovoluje nastavit zpoždění po každé provedené akci pomocí metody `setExecutionDelay`. Je tedy možné spustit test ve zpomaleném módu, pokud je to potřebné. Po dokončení testu je vygenerován report, který přehledně informuje o úspěšnosti dokončených testů.

Mezi zajímavé funkce RiaTest lze zařadit podporu spouštění testů napříč jednotlivými prohlížeči. V rámci jednoho testu lze test spustit na podporovaných prohlížečích, mezi které se řadí Mozilla Firefox, Google Chrome a Internet Explorer.

Dokumentaci ke skriptovacímu jazyku RiaScript lze nalézt na internetové adrese `http://www.cogitek.com/onlinedocs/v6/RIAScript/RIAScriptReference.html`.

Na obr. 14 je prostředí RiaTest. V rámci seznámení se s programem byl vytvořen jednoduchý test, který na základě zadaného počtu opakovaní a nastavených hodnot pracuje s aplikací Kalkulačka.



Obrázek 14: Ukázka testu v RiaTest

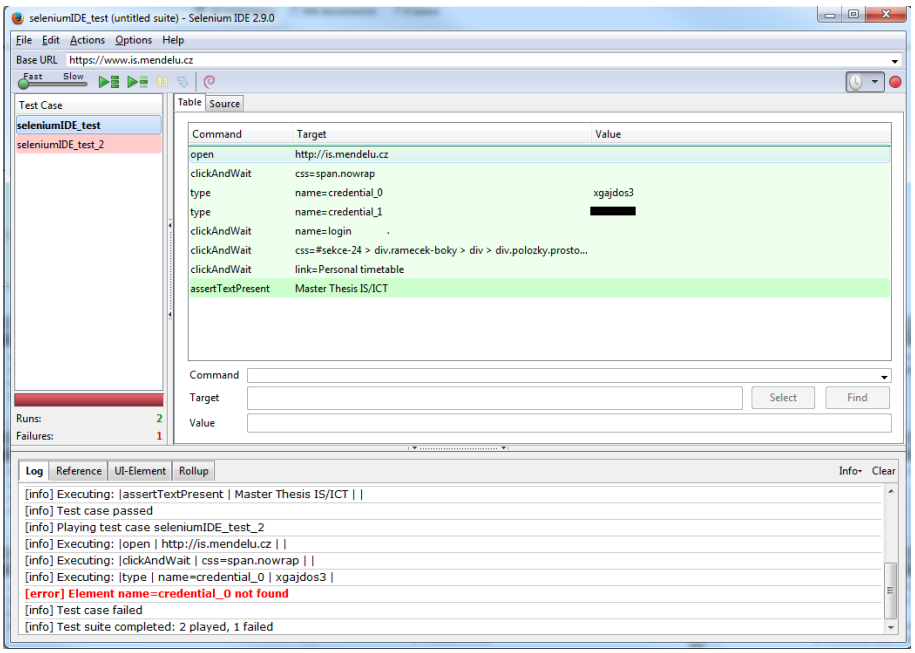
Selenium IDE

Selenium IDE je doplněk do prohlížeče Mozilla Firefox, který je poskytován zdarma. Pomocí Selenium IDE lze vytvořit jednoduché automatické testy pro webové aplikace, internetové stránky a simulaci činností v internetovém prohlížeči. Po spuštění je k dispozici hlavní obrazovka. Na levé straně se nachází seznam všech vytvořených testovacích případů. V prostřední části je prostor pro zadávání a úpravu jednotlivých testovacích kroků. Jednotlivé testovací kroky jsou definovány pomocí příkazů. Příkazy podporují akce jako kliknutí, vložení textu, ověření na výskyt prvku apod. Po vytvoření testovacího případu lze spustit vybrané nebo všechny testovací případy. Před spuštěním lze nastavit rychlost provádění testů. Po dokončení je v dolní části výsledek testovacích případů. Testy lze spouštět ručně nebo je lze naplánovat na zvolený čas. Testy jsou uloženy ve formátu HTML. Jejich obsah lze zobrazit a upravit v záložce `Source`.

Testovací kroky lze psát manuálně nebo použít nahrávání. Při manuálním vytvoření je potřeba vložit nový příkaz s určitým cílem a případnou hodnotou. Užitečná funkce se skrývá pod tlačítkem `Select`, která po najetí na požadovaný prvek v prohlížeči doplní jeho identifikátor do pole `Target`. V případě vytvoření testovacích kroků pomocí nahrávání je každý provedený krok v prohlížeči zaznamenán jako testovací krok. Po skončení nahrávání lze jednotlivé kroky upravit, případně vymazat.

Hlavní výhoda nástroje Selenium IDE je jeho snadná tvorba jednoduchých testů. Mezi nevýhody lze zařadit nevhodnost použití pro složitější testy, jejichž vytvoření a udržování by bylo příliš náročné. Testy lze vytvářet pouze v prohlížeči Mozilla Firefox.

Dokumentace k Selenium IDE je dostupná na internetové adrese `http://docs.seleniumhq.org/docs/index.jsp`.



Obrázek 15: Ukázka testu v nástroji Selenium IDE - kontrola přítomnosti předmětu v osobním rozvrhu

Sikuli

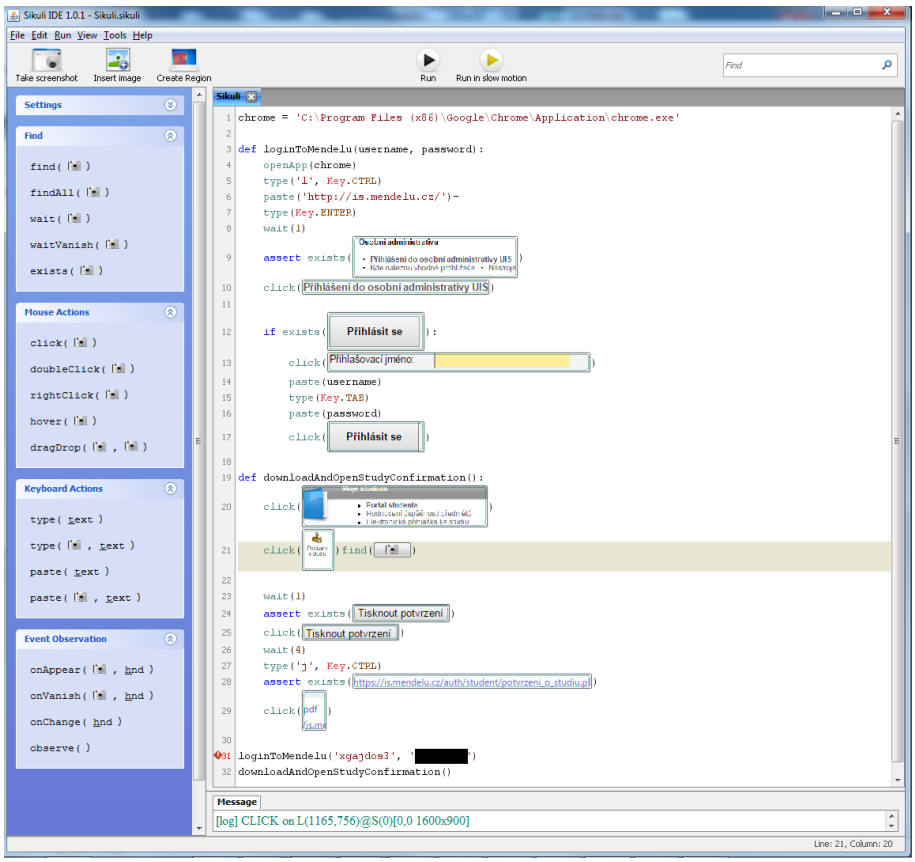
Sikuli je volně dostupný nástroj, za jehož pomoci lze provést automatizaci takřka všeho, co lze vidět na obrazovce. V Sikuli lze implementovat testy za pomocí jazyka Python. Sikuli pro práci s jednotlivými testovanými objekty využívá rozpoznávání obrázků pomocí knihovny OpenCV. OpenCV je volně dostupná knihovna pro manipulaci s obrazem. Díky tomu lze vytvářet testy velmi rychle a efektivně.

Ve vývojovém prostředí Sikuli se na levé straně nachází základní metody pro rychlý přístup, které jsou rozděleny podle určité funkcionality. Jsou zde zastoupeny metody, které spadají pod rozpoznávání, simulace myši a klávesnice. Například pomocí metody `exists()` lze zjistit, jestli zkoumaná hodnota existuje. Pokud je před metodu `exists()` přidáno klíčové slovo `assert`, v případě nenalezení požadované hodnoty je ukončen test a zaznamenán krok, ve kterém požadovaná hodnota nebyla nalezena. Testy lze spouštět ve zpomalené rychlosti, což lze využít například při odladňování.

Pro potřeby automatizace oblastí, které se příliš často nemění nebo je obtížné je automatizovat pomocí jiných nástrojů, je Sikuli užitečný nástroj, zejména pro vytváření samostatných na sobě nezávislých testů. V případě rozsáhlých testů je ale Sikuli méně vhodný. Udržovatelnost většího množství testů je obtížná. Pokud je test spuštěn na obrazovce s odlišným rozlišením, například na obrazovce přenosného počítače, může se stát, že jednotlivé obrázky nejsou korektně rozpoznány. Tato skutečnost se jeví jako největší nevýhoda nástroje Sikuli. Dokumentace k Sikuli je dostupná na internetové adrese

`http://doc.sikuli.org`.

Na obr. 16 je zobrazen příklad testu v nástroji Sikuli. Na základě zadaného uživatelského jména a hesla je uskutečněno přihlášení do osobní administrativy UIS. Následně je provedeno stažení souboru potvrzení o studiu a jeho otevření.



Obrázek 16: Ukázka testu v nástroji Sikuli

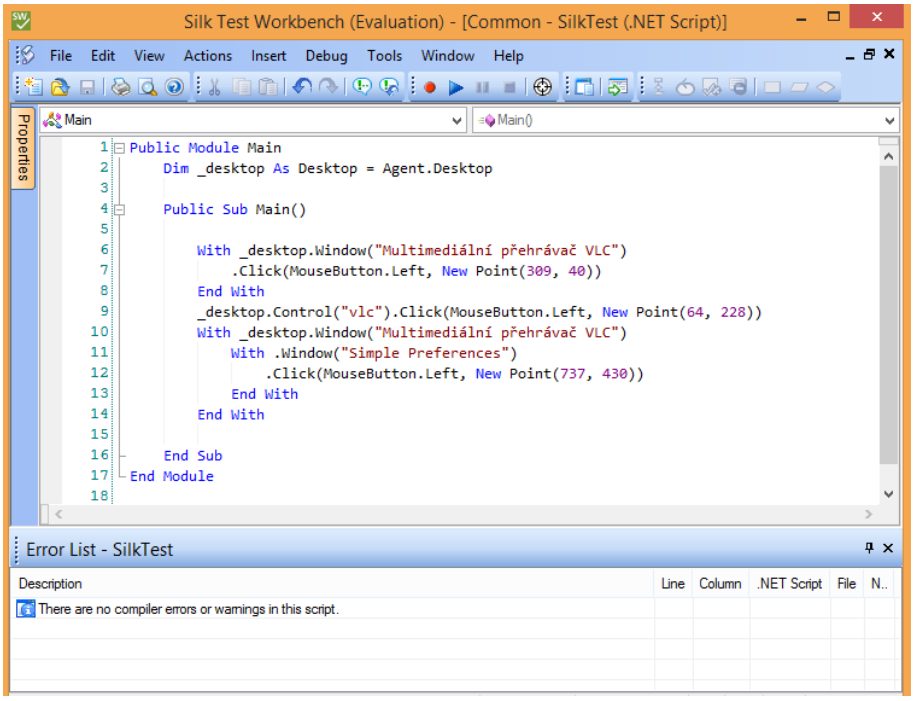
SilkTest

SilkTest se řadí mezi komerční nástroj od společnosti Borland. Jedná se o robustní nástroj, který podporuje mnoho funkcí, které jsou nápomocné v procesu automatizace testování. SilkTest podporuje programovací jazyk Visual Basic .NET.

Pro vytváření testů je možné použít i jiné vývojové prostředí. Testy je možno vytvářet ve vývojovém prostředí Eclipse v programovacím jazyce Java za pomocí doplňku Silk4J. Doplněk Silk4NET poté dovoluje využívat vývojové prostředí Microsoft Visual Studio a psát testy v programovacím jazyce Visual Basic .NET nebo C.

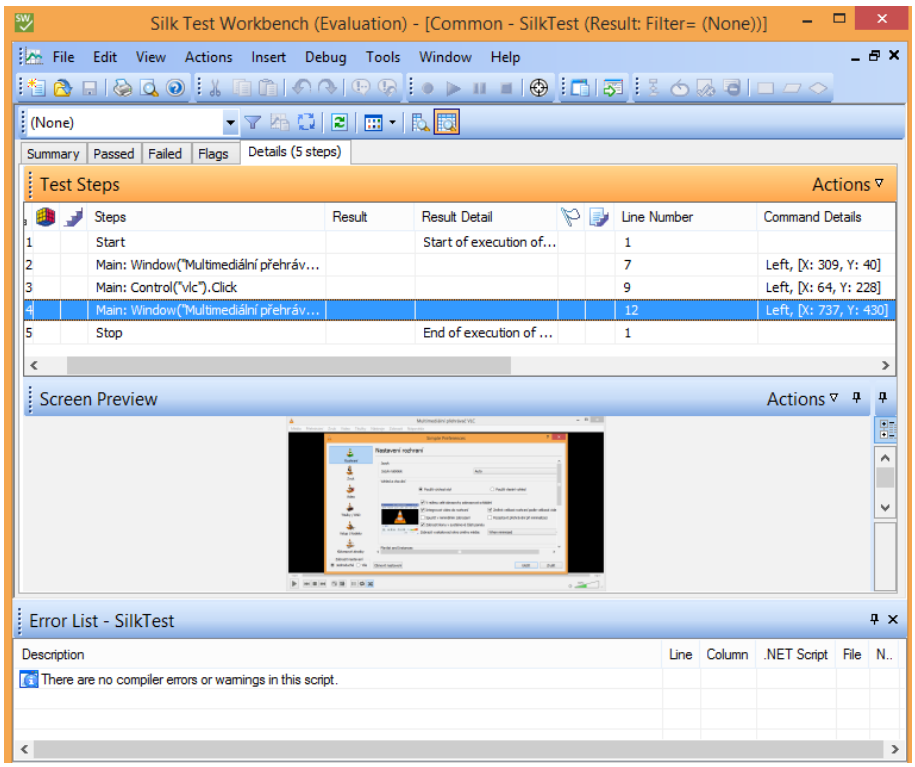
SilkTest podporuje obrazové a textové rozpoznávání, což umožňuje pohodlnější pokrytí při testech. Testy pro webové aplikace lze testovat napříč jednotlivými internetovými prohlížeči. Mezi podporované prohlížeče patří Mozilla Firefox, Google Chrome a Internet Explorer. Při implementaci testů je možné využít podpůrného nástroje pro zaznamenání provedených akcí, které lze poté upravovat. Rozsáhlá uživatelská dokumentace je dostupná na internetové adrese `http://documentation.microfocus.com/help/index.jsp?nav=%2F5_2`.

Na obr. 17 je demonstrace jednoduchého testu, který otevře a zavře okno Nastavení v programu VLC Media Player.



Obrázek 17: Ukázka testu v nástroji SilkTest

Po spuštění a dokončení testu je vygenerována zpráva o průběhu testování. Ta obsahuje základní informace o průběhu testování. Obsahuje také přehled všech kroků, které byly v rámci testu vykonány. U každého kroku je uveden odpovídající řádek v kódu a snímek obrazovky z testu.



Obrázek 18: Zpráva o výsledku testování v nástroji SilkTest

TestComplete

TestComplete je komerční nástroj od společnosti SmartBear Software. TestComplete umožňuje vytvářet automatizované testy pro rozličné typy aplikací v rámci následujících modulů:

- **TestComplete Desktop** – desktopové aplikace,
- **TestComplete Mobile** – mobilní aplikace,
- **TestComplete Web** – webové aplikace.

TestComplete nabízí solidně propracovanou strukturu při práci s jednotlivými testy. Testy lze vytvářet pomocí skriptovacího jazyka, záznamu akce uživatele nebo kombinací obou možností. TestComplete umožňuje implementovat testy v následujících skriptovacích jazycích:

- C++Script,
- CScript,
- DelphiScript,
- JScript,

- VBScript.

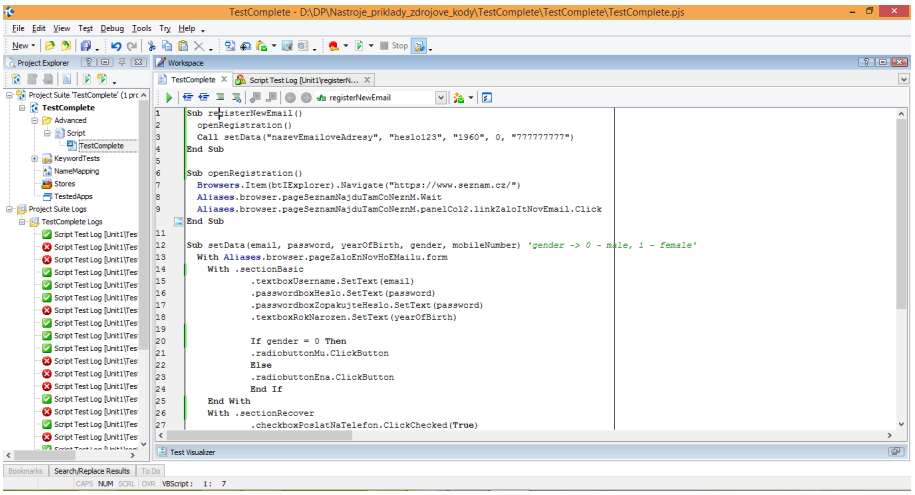
TestComplete podporuje odladění jednotlivých skriptů pomocí vytváření kontrolních bodů ve vývojovém prostředí. Již spuštěné testy lze zastavit, upravit kód a znovu pustit.

Při vytváření testů pomocí záznamu a jejich následné údržbě je k dispozici funkce `Test Visualizer`. Pro každou akci v kódu je přiřazen odpovídající snímek obrazovky, na kterém je zvýrazněn testovaný prvek. Snímek obrazovky může být v závislosti na typu zaznamenané aplikace interaktivní, po najetí na jakýkoli prvek se provede zvýraznění prvku a je zobrazena jeho identifikace.

V rámci TestComplete lze kombinovat provádění manuálních a automatických testů. Po dokončení testu je vygenerována zpráva o výsledku testování. TestComplete lze integrovat s nástroji pro reportování chyb (JIRA a BugZilla), což dovoluje pohodlné vytvoření chyby.

Uživatelská dokumentace je dostupná na internetové adrese `http://goo.gl/7685X4`.

Na obr. 19 je zobrazeno vývojové prostředí a jednoduchý test, který na základě definovaných údajů založí nový email na internetové stránce `http://seznam.cz`.



Obrázek 19: Ukázka testu v nástroji TestComplete

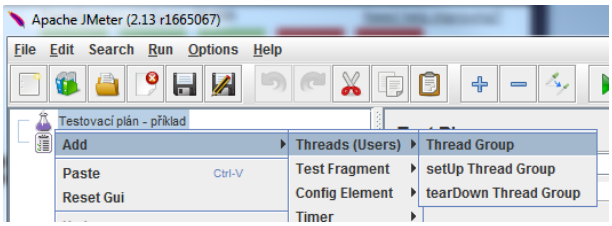
3.4 Nástroje pro výkonnostní testování

V následujících řádcích budou představeny nástroje pro výkonnostní testování a budou vyzkoušeny jejich základní funkce. V rámci jednotlivých nástrojů budou vytvořeny příklady, které demonstrují možnosti zkoumaných nástrojů.

Apache JMeter

JMeter je bezplatně poskytovaný nástroj pro testování aplikace pod vytvořenou zátěží a pro měření její výkonnosti. JMeter zasílá na testovanou webovou aplikaci požadavky a měří, za jakou dobu je požadavek vyřízen. Pro získání důvěryhodných výsledků je doporučeno nespouštět JMeter na stejném stroji jako testovanou aplikaci. Výsledky mohou být zkresleny. Testování je ideální provádět na vyhrazené síti, kde je co nejmenší rušivý provoz (Hynar, 2004).

Pro spuštění jednoduchého testu je potřeba vytvořit potřebné kroky v testovacím plánu. Základním krokem je přidání prvku `Thread Group`, který definuje počet jednotlivých vláken představující uživatele. Prvek lze přidat po vyvolání kontextové nabídky v požadovaném testovacím plánu a následném výběru možnosti `Add - Threads(Users) - Thread Group` (viz obr. 21).

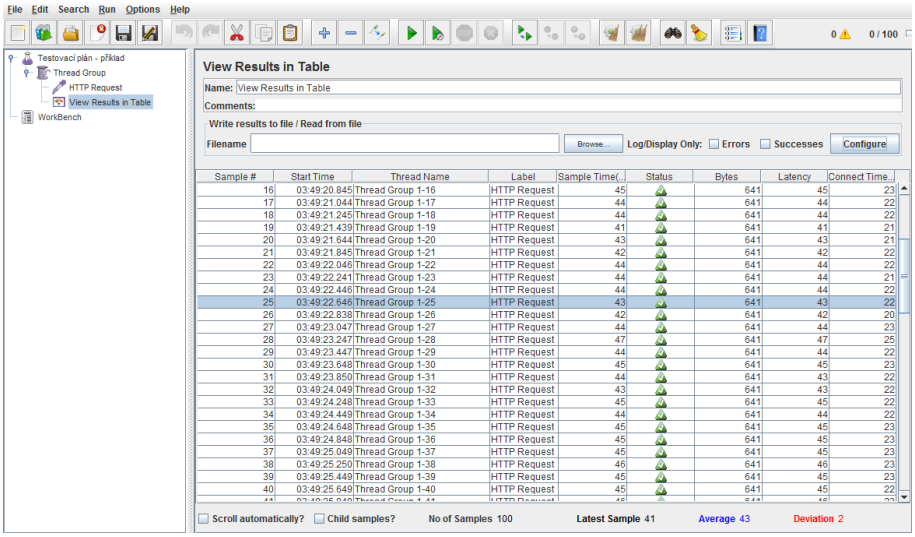


Obrázek 20: JMeter - výběr prvku `Thread Group`

Poté je možné zvolit počet virtuálních uživatelů (`Thread Group`), rychlost prodlevy mezi jednotlivými přístupy každého uživatele v sekundách (`Ramp-Up Period`), v případě zadání nuly je proveden přístup všech uživatelů najednou. Jako další lze zvolit počet opakování (`Loop Count`) a časový plánovač (`Scheduler`). Po zadání požadovaných údajů, lze přidat testování HTTP požadavku. Přidání se provede vyvoláním kontextové nabídky u `Thread Group` a zvolení možnosti `Add - Sampler - HTTP Request`. Do pole `Server Name or IP` lze zadat název nebo IP adresu serveru. V rámci `HTTP Request` lze zadat velké množství dodatečných možností, které se mají vykonat.

Pro zobrazení výstupu z testu je potřeba přidat prvek, který se stará o vizualizaci výsledků. Vyvoláním kontextové nabídky u testovacího plánu lze vybrat například `Add - Listener - View Results in Table`. Poté lze již spustit test. Po dokončení lze zobrazit výsledky, které lze poté uložit do souboru.

Uživatelský manuál je pro Apache JMeter dostupný na internetové adrese `http://jmeter.apache.org/usermanual/index.html`.



Obrázek 21: JMeter - vyhodnocení testu

Silk Performer

Silk Performer je komerční nástroj od společnosti Borland. Podporuje velkou škálu testovatelných technologií. Mezi základní webové technologie, které je možné podrobit zátěžovému testování, patří zejména:

- AJAX,
- Adobe Flash,
- Adobe Flex,
- HTML5,
- Microsoft Silverlight.

Výkonnostní testování je možné provést u mobilních aplikací s operačním systémem:

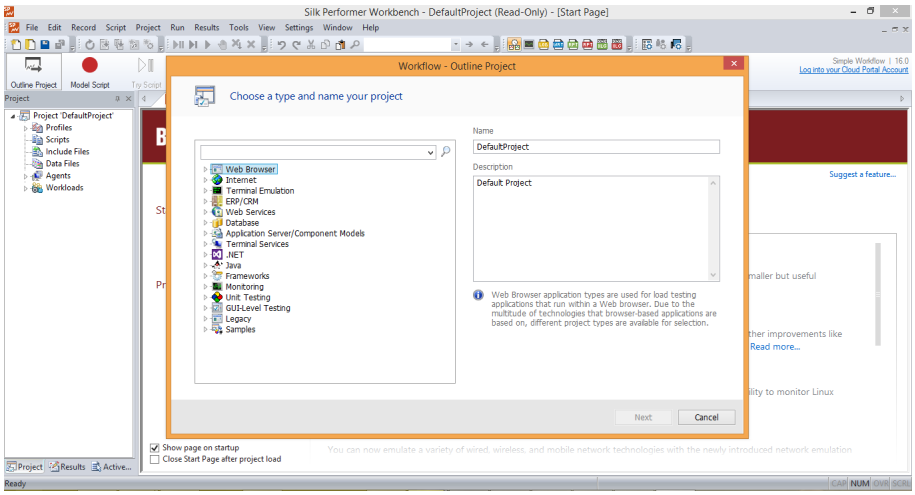
- Android,
- BlackBerry,
- iOS,
- Windows Phone.

U mobilních aplikací lze stejně jako v případě webového prostředí provádět testy, které simulují rozdílné standardy pro mobilní připojení. Podporovány jsou následující:

- EDGE,

- GPRS,
- HSDPA,
- HSPA+,
- LTE,
- UMTS.

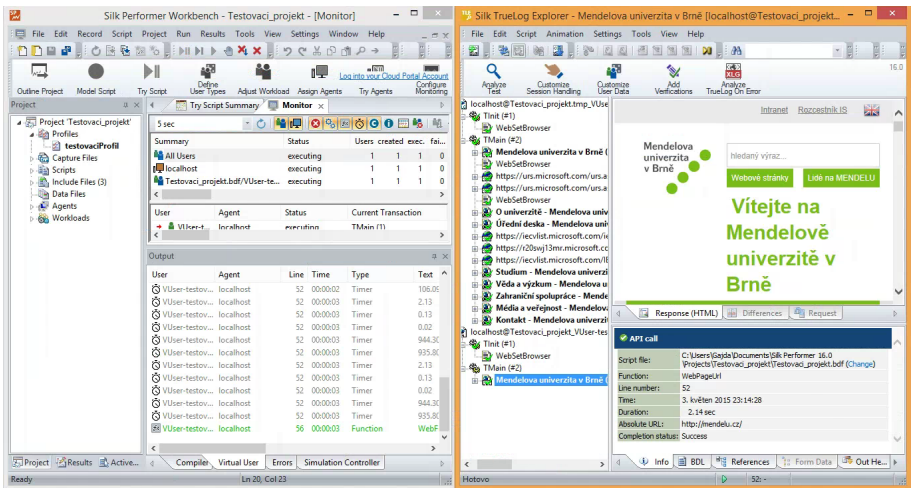
Při vytvoření nového projektu je zvolen výběr technologie, pro kterou bude vytvořen zátěžový test. Na obr. 22 je přehled dostupných typů, které lze v rámci projektu vybrat.



Obrázek 22: Dostupné technologie pro testování v nástroji SilkPerformer

Pro ukázku zátěžového testu byl vybrán test webové stránky. Po vytvoření projektu lze pomocí volby `Model Script` otevřít okno s nastavením, kde lze zvolit požadovaný internetový prohlížeč a internetovou adresu. Možností `Start recording` je spuštěn internetový prohlížeč se zadanou internetovou adresou. Veškeré kroky v prohlížeči jsou zaznamenávány. Po skončení záznamu je vygenerován skript, který lze upravit. V menu `Settings - Active Profile` lze nastavit veškeré parametry pro prováděný test. V této části lze nastavit požadovaný počet virtuálních uživatelů, kteří budou simulovat chování definované ve skriptu. Dále lze nastavit, internetový prohlížeč, pomocí kterého budou virtuální uživatelé přistupovat. Na obr. **??** je zobrazena sekce `Internet`, kde je umístěno nastavení typu připojení, přenosové rychlosti apod.

Po provedení nastavení a potvrzení provedených změn lze prostřednictvím `Try script` spustit vytvořený test. Po dokončení jsou zobrazeny podrobné výsledky o průběhu testování. Na obr. 23 je demonstrován průběh testování internetové stránky `http://mendelu.cz`. Silk Performer je nástroj, který dovoluje velké množství nastavení a možností pro jednotlivé testy v rámci podporovaných technologií.



Obrázek 23: Ukázka probíhajícího zátěžového testu

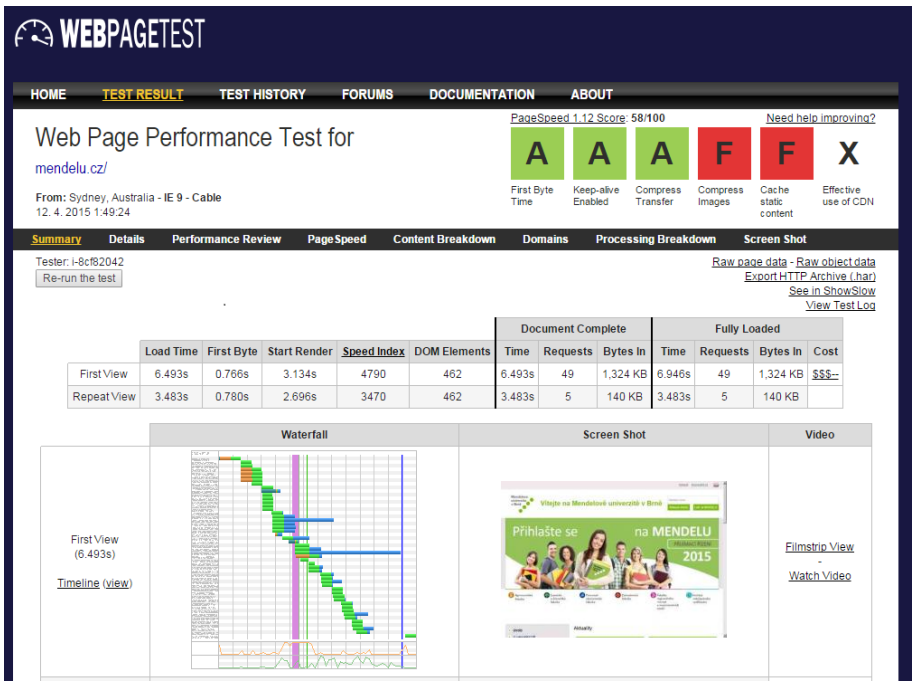
K nástroji Silk Performer je k dispozici podrobná uživatelská dokumentace na internetové adrese `http://goo.gl/9FufEX`.

WebPagetest

WebPagetest je webová aplikace, která je dostupná zdarma na internetové stránce `http://www.webpagetest.org`. Webpagetest umožňuje provedení podrobného testu zadané internetové stránky. Ten se zaměřuje zejména na rychlost načítané stránky.

Na úvodní stránce je možnost zadat internetovou stránku, která bude testována. Poté lze určit, z jaké lokality bude simulován přístup virtuálních uživatelů a jaký internetový prohlížeč bude pro test použit. WebPagetest dovoluje určit i požadovanou rychlost připojení a podporuje i internetové stránky, které jsou dostupné pouze po zadání přihlašovacích údajů. Po dokončení testu je zobrazen podrobný výsledek, který obsahuje čas, za který se načetla stránka poprvé a následně opakovaně. Součástí výsledku je i ohodnocení jednotlivých oblasti formou známek. Po kliknutí na určitou známku jsou zobrazeny problematické oblasti a návrh jejich zlepšení. Ve výsledku je zaznamenán i videozáznam testu. Ten je možné přehrát a lze tak zhlédnout, jak se stránka načítala z uživatelského pohledu.

Na obr. 24 je zobrazen výsledek testu pro `http://mendelu.cz`. Pro test byl použit přístup ze Sydney.



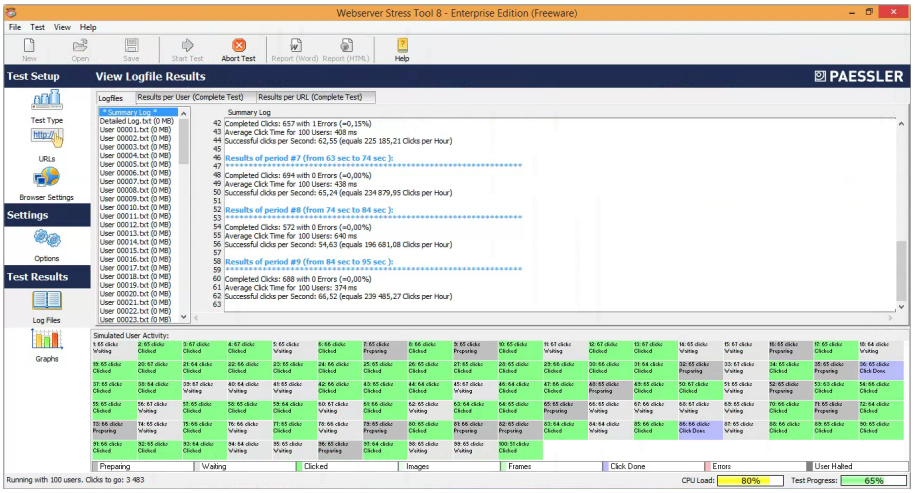
Obrázek 24: Ukázka výsledku testu v nástroji WebPagetest

Webserver Stress Tool

Webserver Stress Tool je bezplatně poskytovaný nástroj od společnosti Paessler určený pro výkonnostní testování webového serveru. Představuje jednoduchý a intuitivní nástroj pro vytvoření základních zátěžových testů. Aplikace je rozdělena do 3 základních sekcí:

- **Test Setup** – zde se nachází nastavení typu testů. Test může byt nastaven buď v závislosti na počtu kliknutí nebo na nastaveném čase, na základě kterého se běh testu provádí v neměnné nebo stoupající zátěži. Jako další kritérium lze nastavit počet virtuálních uživatelů a zpoždění prováděných simulovaných kliknutí. Následovně je možné zadat sekvenci URL adres, které budou testovány. Pro simulaci přístupu virtuálních uživatelů je umožněno vybrat internetový prohlížeč.
- **Settings** – zde je umístěno rozšířené nastavení včetně časového plánovače pro spouštění testů.
- **Test Results** – v části `Test Results` jsou uloženy informace z průběhu testování a po dokončení jsou vygenerovány i výsledky ve formě grafu. Výsledky testování je umožněné exportovat do formátu `DOC` nebo `HTML`.

Na obr. 25 je znázorněn průběh zátěžového testu internetové stránky `http://mendelu.cz`. Přehledný uživatelský manuál je zpřístupněn na internetové adrese `http://download-cdn.paessler.com/download/webstressmanual.pdf`.



Obrázek 25: Ukázka probíhajícího testu se 100 virtuálními uživateli

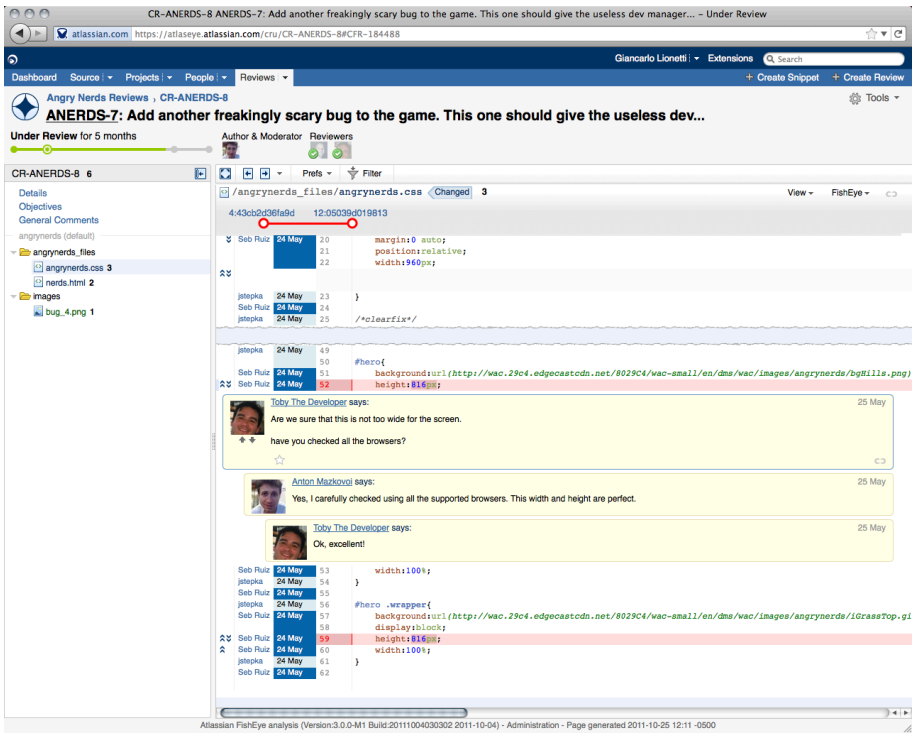
3.5 Nástroje pro revizi a kontrolu kódu

V následující podkapitole budou představeny některé z nástrojů, které lze použít pro revizi a kontrolu kódu.

Crucible

Crucible je komerční nástroj od společnosti Atlassian. Crucible spadá do skupiny nástrojů, které jsou určeny ke kontrole vytvořeného nebo upraveného kódu. Po provedení změny v kódu je v nástroji Crucible založené tzv. `Code review`, které je přiřazeno pro kontrolu určité osobě. `Code review` obsahuje seznam jednotlivých zdrojových souborů, které byly změněny. Po otevření detailu jednotlivých souborů jsou barevně zvýrazněny provedené změny. Osoba, provádějící kontrolu, poté může přidat komentář k problematickým částem kódu, upozornit na možnost řešení již existující metodou apod.

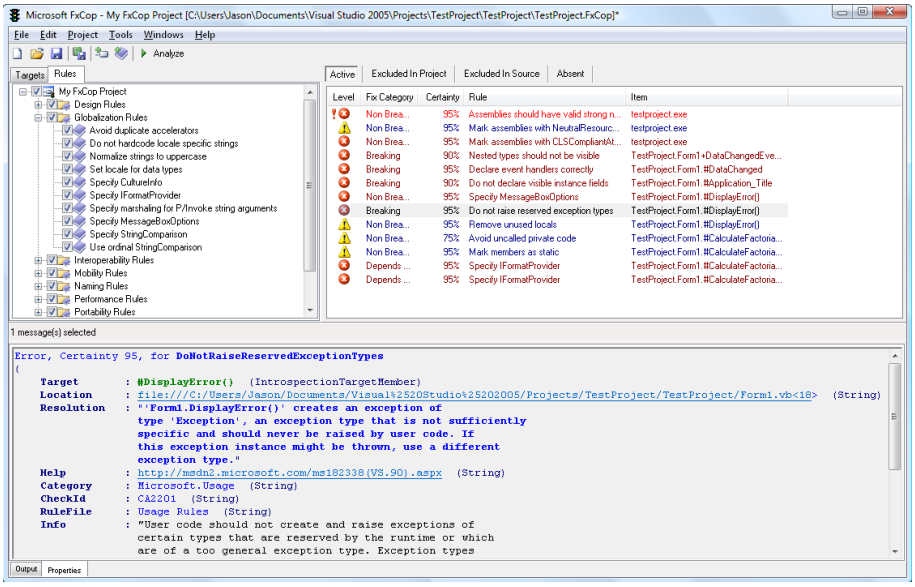
Smyslem provádění `Code review` je zejména snaha o vyšší kvalitu kódu. Skutečnost, že implementovaný kód bude kontrolován jinou osobou, vede ke psaní čitelnějšího, lépe dokumentovaného kódu. Výhodou je i předávání znalostí mezi jednotlivými účastníky. V neposlední řadě je tato praktika vhodná pro zaučení nově příchozích zaměstnanců. Na obr. 26 je ukázka z prostředí Crucible.



Obrázek 26: Ukázka `Code review` v nástroji Crucible (Atlassian, 2015)

FxCop

FxCop je nástroj od společnosti Microsoft, který dokáže analyzovat upravovaný kód a poskytne řadu informací. Mezi tyto informace lze zařadit například doporučení pro možné vylepšení návrhu, výkonu, bezpečnosti apod. FxCop dokáže detekovat řadu běžných programátorských chyb. FxCop je k dispozici jako samostatný nástroj a je zahrnut v prostředí Visual Studio. Na obr. 27 je zobrazen nástroj FxCop (Page, Johnston a Rollison, 2008).



Obrázek 27: Ukázka nástroje FxCop (BinaryCoder)

4 Vlastní práce

V následujících podkapitolách budou popsány požadavky na vytvoření automatizovaných testů, návrh testů, metodika tvorby automatizovaných testů a výsledný stav.

4.1 Požadavky na automatizaci

Automatizované testy je potřebné implementovat pro rozsáhlou webovou službu v rámci konkrétní společnosti. Webová služba je definována pomocí `WSDL`. `WSDL` (Web Sevices Description Language) je jazyk určený k popisu funkcionality určité webové služby. Používá se k vytváření nebo generování souboru `.wsdl`. Ten poté mohou další služby použít k určení vystavené funkčnosti nabízené webovou službou. `SOAP` (Simple Object Access Protocol) se používá jako základní formát komunikace mezi webovými službami. Funkčně používá `SOAP` k popisu svého obsahu jazyk `XML` (Microsoft ACE Team, 2003).

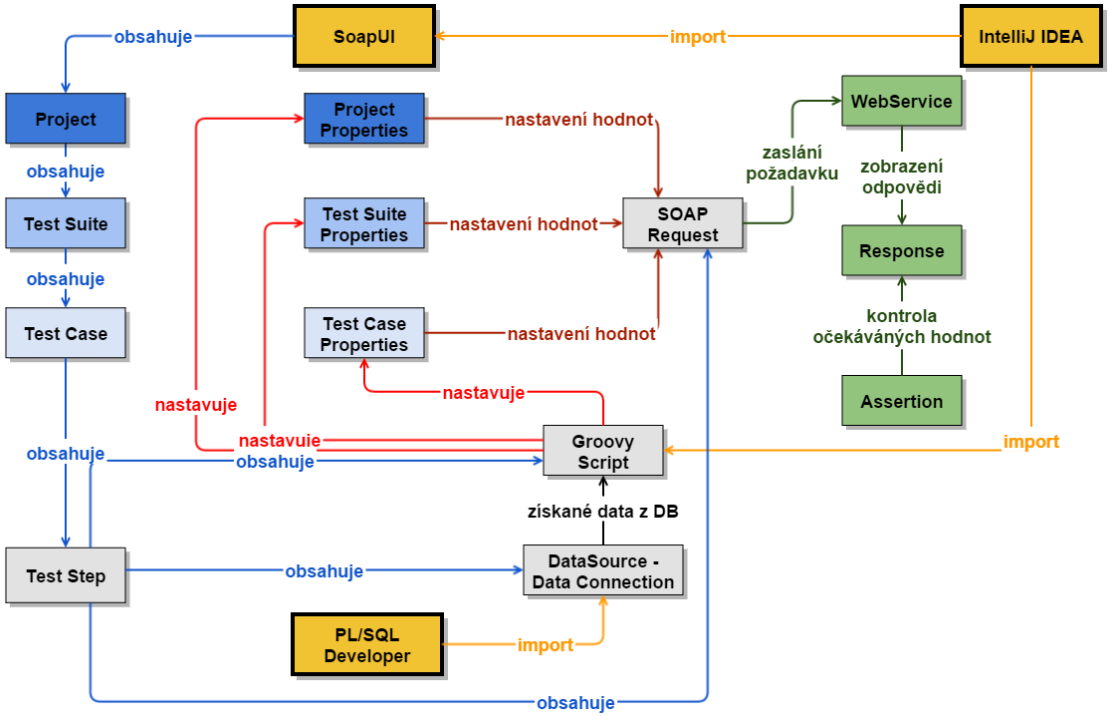
Pro otestování webové služby, pro kterou je potřebné vytvořit automatizované testy, je v rámci jednoho požadavku (`Request`) odeslaného na adresu webové služby (Endpoint) vyžadováno zadání cca 60 hodnot, které splňují určité kritéria. Po odeslání je zobrazena odpověď na požadavek (`Response`). Současně jsou vytvořeny záznamy v databázi, které je potřeba kontrolovat. Na základě zadaných hodnot v požadavku je možné dosáhnout několika desítek průběžných a konečných stavů. Dosažení jednotlivých stavů a průchodů je definováno interní strategií. Jelikož se jedná o kritickou funkcionalitu systému a vygenerované data jsou prerekvizitou pro další testování, je zapotřebí tuto činnost provádět opakovaně a různých typech prostředí (vývojová, testovací, integrační apod.). V případě manuálního testování by byl proces popsaný výše velmi neefektivní, časově velmi náročný a náchylný k chybám způsobených například přehlédnutím, únavou osoby provádějící testy apod. Taktéž následná kontrola v databázi, která zahrnuje ověření dat u každého jednotlivého zaslaného požadavku oproti očekávanému výsledku, by v případě manuálního provedení zabrala neadekvátní množství testovacích kapacit.

Z důvodu práce s citlivými údaji společnosti budou obrázky z programu SoapUI na některých místech upraveny, jelikož zobrazují vnitřní strukturu, definice webové služby apod.

4.2 Návrh testů

Na základě požadavků pro automatizování testů bylo potřeba určit, jaké nástroje budou vhodné pro vytvoření požadované sady testů. Jedno z hlavních kritérií byla již výše zmíněná možnost opakovaně spouštět testy. Bylo také potřebné analyzovat veškerá vstupní data. Pro nasimulování velkého množství požadovaných průběžných a konečných stavů bylo potřeba prozkoumat, na základě jakých podmínek je možné dosáhnout požadovaných stavů. Na základě těchto poznatků proběhl výběr potřeb-

ných nástrojů. Následně byl vytvořen návrh, který definuje postup při vytvoření jednotlivých testů. Na obr. 28 je demonstrován tento základní návrh.



Obrázek 28: Základní návrh tvorby testů

4.3 Použité nástroje

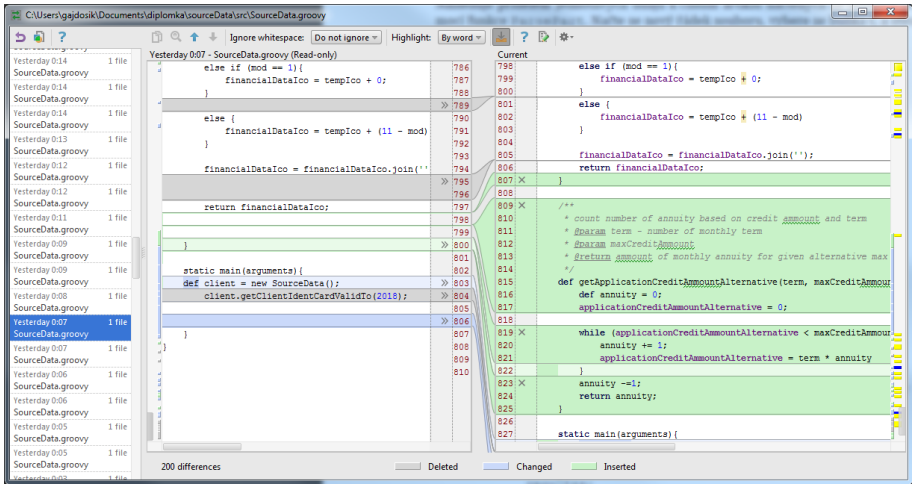
IntelliJ IDEA

IntelliJ IDEA je vývojové prostředí od společnosti JetBrains. V základní verzi je poskytována zdarma a podporuje vývoj v následujících technologiích:

- Android,
- Groovy,
- Java,
- Scala.

IntelliJ Idea má uživatelsky přívětivé prostředí. Součástí jsou nástroje, které usnadňují samotný proces tvorby kódu. Jedná se zejména o našeptávání existujících proměnných, metod, tříd apod. IntelliJ IDEA má propracovaný systém práce s historií implementovaného kódu. V závislosti na nastavení jsou do historie ukládány jednotlivé verze. U každé jednotlivé verze je graficky znázorněno, která část kódu byla

změněna, přidána nebo odstraněna. Zobrazuje také celkový počet provedených změn. Historie je přístupná přes `VCS - Local History - Show History`. Na obr. 29 je zobrazena ukázka práce s historií.

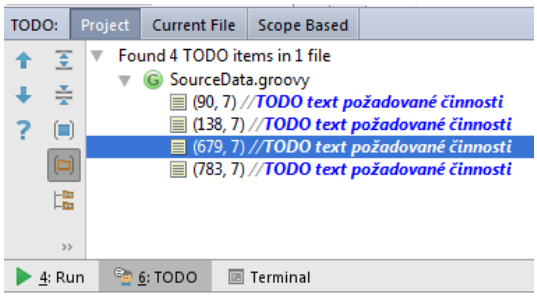


Obrázek 29: Ukázka historie v nástroji IntelliJ IDEA

Mezi další podpůrné funkce patří podpora pro generování dokumentace v rámci nástroje Javadoc a GroovyDoc. Po zadání `/**` v části nad určitou metodou a potvrzení klávesou `Enter` se automaticky doplní šablona, kde lze zadat účel metody, popis vstupních hodnot a hodnotu, která je vrácena. Poté lze jednoduše vygenerovat dokumentaci v přehledném `HTML` formátu. Níže je demonstrována ukázka šablony a použití.

```
/**
  * popis metody
  * @param hodnota - popis vstupní hodnoty
  * @return - popis výsledku
  */
 def priklad(hodnota){
     return hodnota;
 }
```

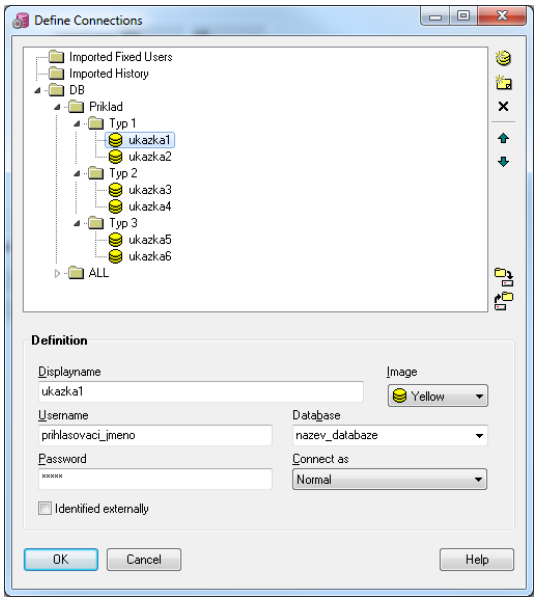
IntelliJ IDEA disponuje užitečnou vlastností, která je vhodná v případě, pokud je potřeba některou část kódu v budoucnu vylepšit nebo upravit. Do požadované části kódu se do těla komentáře zapíše příkaz `TODO`, například tedy `//TODO text požadované činnosti`. Poté je ve spodní části pod záložkou `TODO` dostupný seznam všech pozicí v kódu, kde byl použit příkaz `TODO`. Po kliknutí na určitý prvek je provedeno přesunutí přímo na odpovídající místo. Na obr. 30 je ukázka přehledu `TODO` prvků.



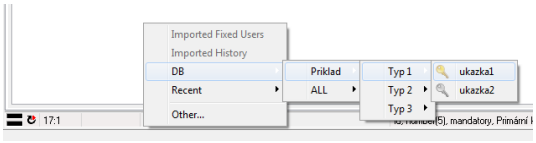
Obrázek 30: Ukázka `TODO` prvků v nástroji IntelliJ IDEA

PL/SQL Developer

PL/SQL Developer je komerční nástroj od společnosti Allround Automations. V rámci tohoto nástroje lze komplexně zpracovávat požadavky v databázové oblasti. Potřebné údaje pro přístup do jednotlivých databází lze definovat v `Tools - Define Connections`. Zde lze nastavit požadované údaje pro korektní připojení na jednotlivé databáze (viz obr. 31). Po dokončení je možné uložené definice exportovat a importovat na jiném počítači. Poté lze již vytvořené definice prostředí používat. Na obr. 32 je zobrazena nabídka připojení na jednotlivé definované databáze. Není tak potřeba neustále zadávat přihlašovací údaje, název databáze apod.

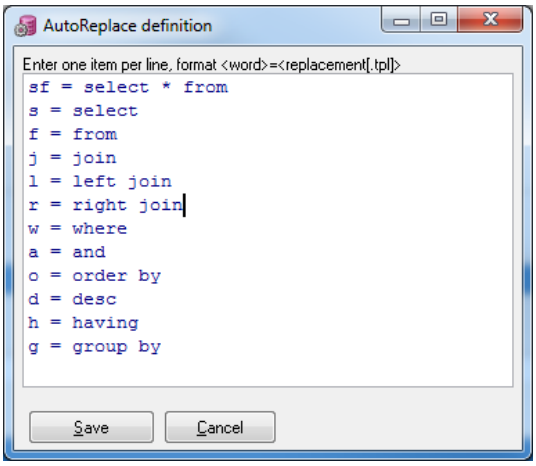


Obrázek 31: Ukázka nastavení připojení v nástroji PL/SQL Developer



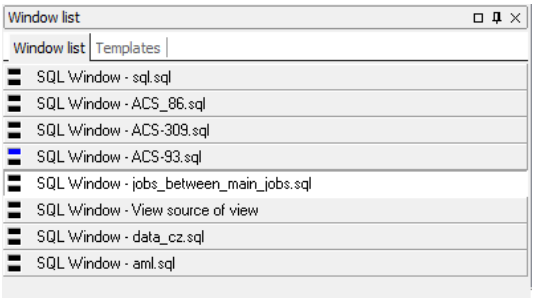
Obrázek 32: Ukázka připojení na definovanou databázi

V PL/SQL Developer nabízí možnost využívání zkratek během psaní kódu, které ulehčí a zrychlí práci. Nastavení je možné provést v `Tools - Preferences - Editor - AutoReplace`. Po volbě `Edit` se zobrazí okno, kde je možné nastavit zkratky a jejich význam. Po dokončení definice je provedeno uložení v textovém souboru. Na obr. 33 je zobrazeno nastavení zkratek.



Obrázek 33: Ukázka funkce `AutoReplace` v nástroji IntelliJ IDEA

Pokud je vyžadováno připojení na několik databází nebo je potřeba pracovat s více okny najednou, lze použít možnost `Tools - Window list`. Ta dokáže zobrazit všechna otevřená okna, přejmenovávat je dle potřeby apod. Na obr. 34 je ukázka `Window list`.



Obrázek 34: Ukázka `Window list` v nástroji PL/SQL Developer

PL/SQL Developer dokáže v průběhu psaní dotazu po stisku klávesy`F6` aktivovat našeptávač, který zobrazí dostupné tabulky apod.

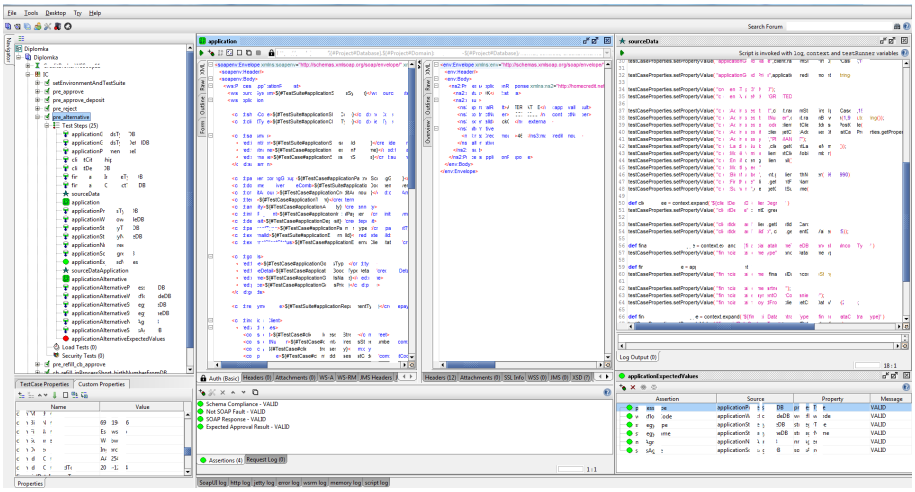
SoapUI

SoapUI je nástroj pro testování funkčnosti webových služeb, který je v základní verzi poskytován zdarma. SoapUI podporuje testování webových služeb v manuálním a automatizovaném režimu.

V SoapUI lze vytvářet automatizované testy. Ty obsahují testovací skupiny (`Test Suite`), které v sobě zahrnují jednotlivé testovací případy (`Test Case`). Testovací případy poté obsahuji jednotlivé kroky (`Test Step`).

Základní testovací krok je reprezentován zasláním dotazu na webovou službu. Jako další testovací kroky mohou být použity dotazy na webové služby, které na sebe mohou vzájemně navazovat a přebírat mezi sebou hodnoty. Dále to můžou být data, která představují výsledek dotazů z databáze, skriptů v jazyce Groovy, používaných metod z importovaných zdrojových souborů v jazyce Groovy apod. Získaná data lze poté nastavit jako zdroj hodnot v rámci jiných testovacích kroků, testovacích případů apod.

Při spuštění testu jsou v závislosti na nastavení postupně spouštěny veškeré jednotlivé kroky v rámci testovacího případu. U testovacího kroku, který zahrnuje dotaz webovou službu je poté obdržena patřičná odpověď. V rámci očekávaného výsledku odpovědi lze na tuto odpověď nastavit kontrolu (`Assertions`), jestli jsou hodnoty dle očekávání. Kontroly lze využít i v jiných případech. Jako testovací krok lze například použít dotaz do databáze. Následně lze vytvořit další testovací krok, který bude kontrolovat správnost získaných údajů z databáze. Spouštět lze jak celé testovací skupiny, tak jednotlivé testovací případy nebo jednotlivé testovací kroky.



Obrázek 35: Ukázka uživatelského rozhraní SoapUI

4.4 Metodika práce

IntelliJ IDEA

V nástroji IntelliJ IDEA byly za pomocí programovacího jazyka Groovy vytvořeny funkce, které zajišťují generování podstatné části dat, které jsou nezbytné pro úspěšné vytvoření automatizovaných testů. Vytvořený soubor se zdrojovým kódem byl následně importován do nástroje SoapUI. Po importování zdrojového kódu lze v SoapUI vytvořit instanci třídy a použít dostupné implementované funkce. Níže je znázorněna jedna z funkcí, která generuje validní IČO - identifikační číslo osoby.

```
def getFinancialDataIco(){
    def tempIco = [];
    def sumTempIco = 0;
    def multipleIco = 8;
    def mod = 0;

    for (i = 0; i < 7; i++){
        tempIco += randomNumber(10);
    }

    for (i = 0; i < tempIco.size(); i++){
        sumTempIco += tempIco[i] * multipleIco;
        multipleIco -= 1;
    }

    mod = sumTempIco.mod(11);
    if (mod == 0 || mod == 10){
        financialDataIco = tempIco + 1;
    }
    else if (mod == 1){
        financialDataIco = tempIco + 0;
    }
    else {
        financialDataIco = tempIco + (11 - mod)
    }

    financialDataIco = financialDataIco.join('');
    return financialDataIco;
}
```

Další ukázkou je funkce generující validní formát rodného čísla. Vstupní hodnoty jsou v intervalu od minimálního do maximálního požadovaného roku.

```
def getClientBirthNumber(min,max){
```

```
        def fullYear = randomBetween(min,max);
        def year = clientBirthNumberYear(fullYear);
        def month = randomBetween(1,13);
        def day =
        randomBetween(1,clientBirthNumberMaxNumberOfDay(month));
        def lastDigits = clientBirthNumberLastDigits(fullYear);
        def complete = year + month + day + lastDigits;
        def identList;

        def mod = complete.mod(11);

        while (mod !=0){
            fullYear = randomBetween(min,max);
            year = clientBirthNumberYear(fullYear);
            month = randomBetween(1,13);
            day =
            randomBetween(1,clientBirthNumberMaxNumberOfDay(month));
            lastDigits = clientBirthNumberLastDigits(fullYear)
            complete = year + month + day + lastDigits;
            mod = complete.mod(11);
        }

        if (month < 10){
            month = '0' + month;
        }
        if (year < 10){
            year = '0' + year;
        }
        if (day < 10){
            day = '0' + day;
        }

        identList = [year,month,day,lastDigits];
        clientBirthNumber = identList.join('');

        return clientBirthNumber;
    }
```

PL/SQL Developer

V PL/SQL Developer byly vytvořeny SQL dotazy pro získání potřebných dat z databáze. Dále zde byly vytvořeny dotazy, které kontrolovaly požadované hodnoty

v databázi po provedeném testovacím případu. Dotazy byly poté použity v nástroji SoapUI.

SoapUI

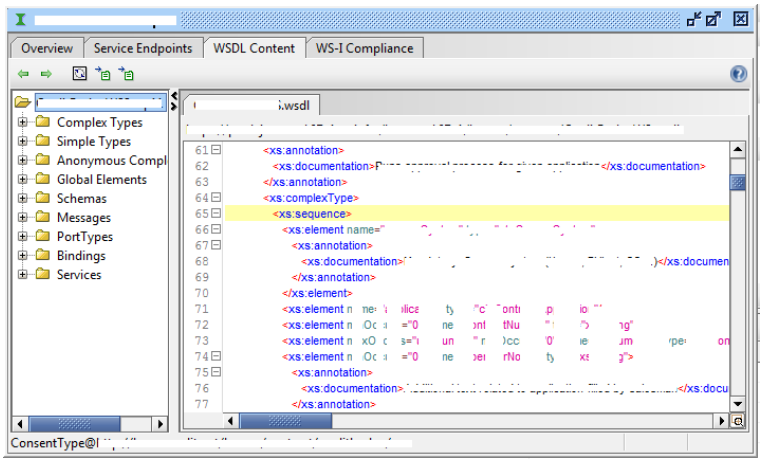
V následujících řádcích bude popsána metodika a jednotlivé kroky při vytváření automatizovaných testů v nástroji SoapUI. Zvolená metodika tvorby testů kladla důraz na opakovatelnost testů a snadné zajištění běhu na rozdílných prostředích.

Vytvoření projektu

Jako první bylo potřeba vytvořit nový projekt. Nový projekt lze vytvořit přes nabídku `File - New Project`. Následně byla vybrána možnost `Create project from - WSDL definition (SOAP)`. Poté byla zvolena cesta k `.wsdl` souboru, který definuje webovou službu. Následně bylo dokončeno vytvoření projektu.

Nastavení projektu

Po vytvoření projektu bylo v rámci projektu vygenerováno rozhraní webové služby - `Interface`. V rámci rozhraní webové služby lze zjistit její obsah. Z obsahu lze analyzovat použité typy hodnot, které mohou být nápomocné při vytváření testů. Na obr. 36 je zobrazen obsah definice webové služby.

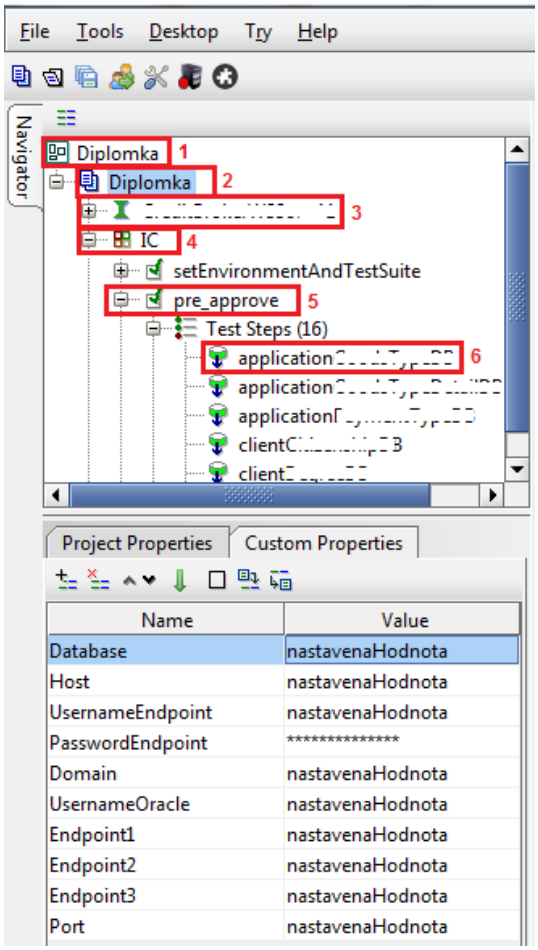


Obrázek 36: Obsah definice webové služby

V rámci nastavení univerzálnosti je vhodné nastavit veškeré údaje, které jsou potřebné pro práci na požadovaném prostředí, na projektové úrovni. Je potřeba definovat požadované proměnné v `ustom Properties` na aktuálním projektu. Po nastavení je možné se na proměnné odkazovat v rámci celého projektu pomocí `${#Project#NazevPromenne}`. Údaje jako název prostředí, databáze, adresa webové služby apod. se poté při změně prostředí nemusí ručně měnit na všech místech, kde se vyskytují, ale jsou přebírány z těchto proměnných. Tím je zajištěna univerzálnost

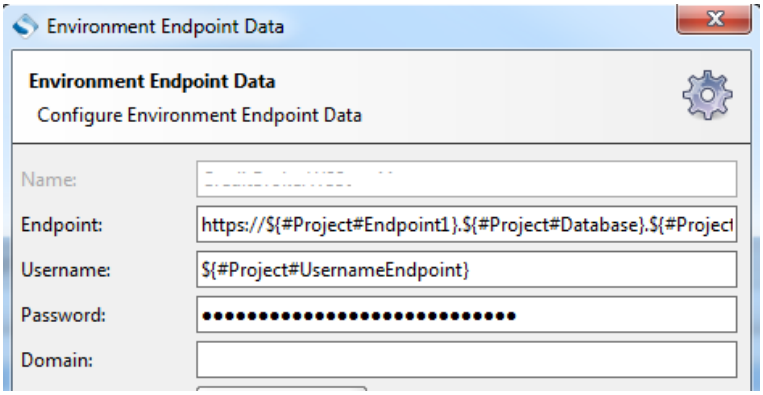
a znovupoužitelnost. Proměnné lze uložit do souboru. Uložené soubory s nastavením lze poté nahrát do dalšího projektu. Proměnné lze nastavit v rámci projektu (`Project`), testovací skupiny (`Test Suite`) a testovacího případu (`Test Case`). Na obr. 37 je znázorněno nastavení proměnných a struktura projektu v rámci SoapUI pomocí číselného označení. Jednotlivá čísla představují následující prvky:

- **1 – Pracovní prostor** (`Workspace`) – slouží jako složka pro jednotlivé projekty.
- **2 – Projekt** (`Projekt`) – slouží pro veškeré nastavení v rámci projektu.
- **3 – Rozhraní** (`Interface`) – obsahuje definici webové služby.
- **4 – Testovací skupina** (`Test Suite`) – obsahuje jednotlivé testovací případy.
- **5 – Testovací případ** (`Test Case`) – obsahuje jednotlivé testovací kroky.
- **6 – Testovací krok** (`Test Step`) – provádí jednotlivé testovací kroky.



Obrázek 37: Struktura projektu v SoapUI

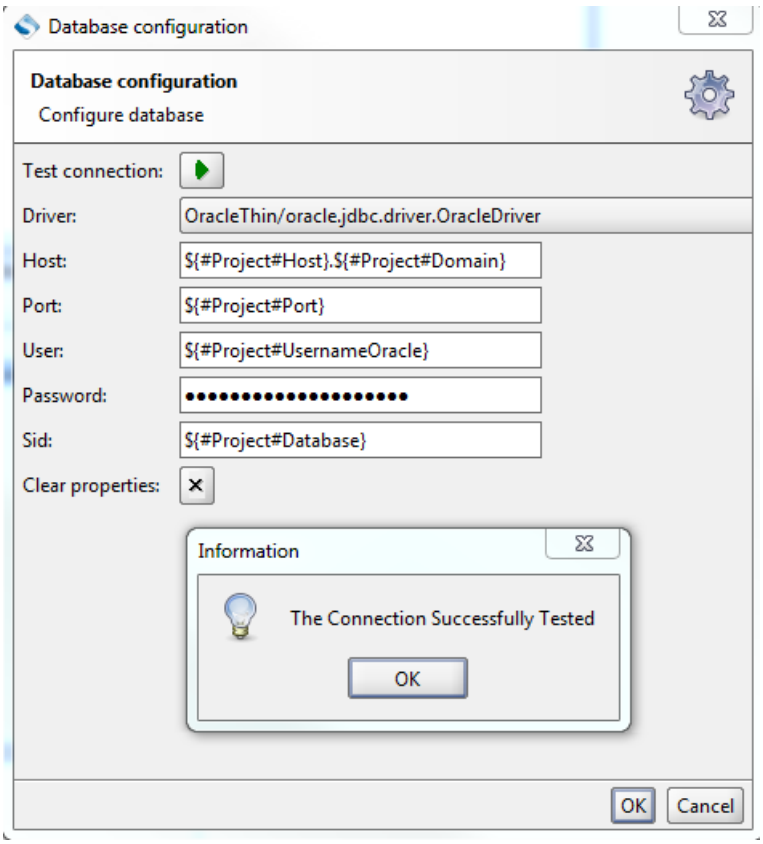
Dalším krokem je nastavení prostředí v rámci projektu. Na záložce `Environments` je možnost přidání nového prostředí. Po vytvoření prostředí je potřebné nastavit adresu webové služby (`Endpoint`). Nastavení je dostupné v záložce `SOAP Services` po označení požadovaného prostředí. Pro nastavení adresy je využito přebírání hodnot z proměnné projektu, stačí tedy nastavit hodnoty pomocí `${#Project#NazevPromenne}` a nemusí se již při každé změně měnit nastavení ručně. Na obr. 39 je znázorněno použití proměnných pro nastavení adresy webové služby.



Obrázek 38: Nastavení adresy webové služby

Následně je potřebné nastavit údaje pro připojení do databáze pomocí `JDBC` (Java Database Connectivity), které umožní práci s databází. Nastavení se nachází v záložce `JDBC Connections` po označení příslušného prostředí. Stejně jako v případě nastavení adresy webové služby jsou hodnoty definovány proměnnými z projektové úrovně. V práci byl pro připojení použit ovladač `OracleThin/oracle.jdbc.driver.OracleDriver`. Pomocí `Test Connection` lze otestovat funkčnost připojení. Pokud pokus o připojení selže a objeví se chybová hláška – `Failed to init connection for driver`, je potřebné přidat příslušný ovladač do adresáře, kde je nainstalován SoapUI – `SoapUI/bin/ext`.

Pro možnost využívání externích zdrojových kódů vytvořených v jazyce Groovy je nezbytné nastavit cestu k souborům se zdrojovým kódem. Nastavení se provede pomocí `File - Preferences - SoapUI Pro`. Do pole `Script Library` se nastaví cesta k požadovanému souboru. Po nastavení je soubor v SoapUI obnovován v intervalu několika málo vteřin. Změny, které se provedou v rámci souboru, se tedy projeví během krátké doby. V rámci testovacího kroku lze poté používat metody z externího souboru. V testovacím kroku (`Groovy Script`) stačí vytvořit instanci požadované třídy (`def nazevInstance = new nazevTridy()`) a poté již používat potřebné metody (`nazevInstance.nazevMetody()`).



Obrázek 39: Nastavení připojení na databázi

Testovací skupina (`Test Suite`)

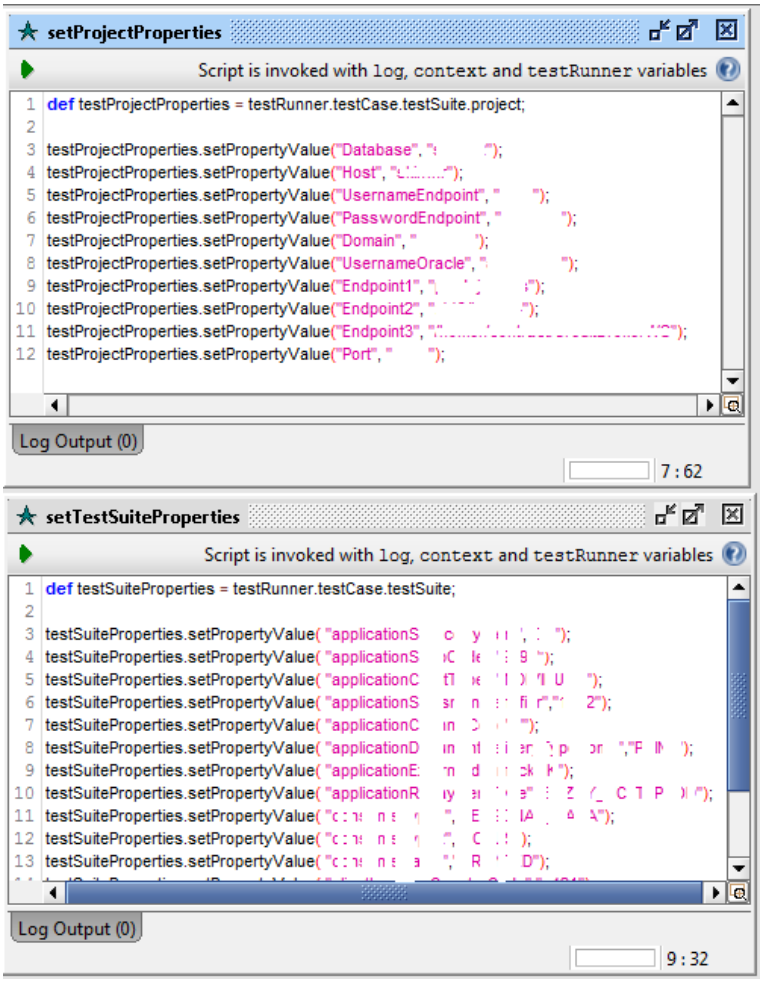
Testovací skupina seskupuje jednotlivé testovací případy. V rámci testovací skupiny je vhodné nastavit do proměnných (`Custom Properties`) hodnoty, které jsou univerzální pro jednotlivé testovací případy nebo je potřeba hromadně měnit jednotlivé hodnoty u všech testovacích případů, které náleží do určité testovací skupiny.

Testovací případ (`Test Case`)

Testovací případ obsahuje jednotlivé testovací kroky. Jako první testovací případ v rámci testovací skupiny byl vytvořen případ, který zajišťuje nastavení proměnných (`Custom Properties`) v rámci projektu a testovací skupiny. Realizován je pomocí dvou testovacích kroků (`Groovy Script`). Nastavení proměnných v projektové úrovni je realizováno pomocí metody `setPropertyValue("nazevPromenne", "hodnota")`. Projektová úroveň je přístupná přes objekt `testRunner.testCase.testSuite.project`.

Nastavení proměnných v úrovni testovací skupiny je realizováno taktéž pomocí metody `setPropertyValue("nazevPromenne", "hodnota")`. Přístupná je přes ob-

jekt `testRunner.testCase.testSuite.project`. Na obr. 40 je znázorněno nastavení proměnných pro projekt a testovací skupinu.

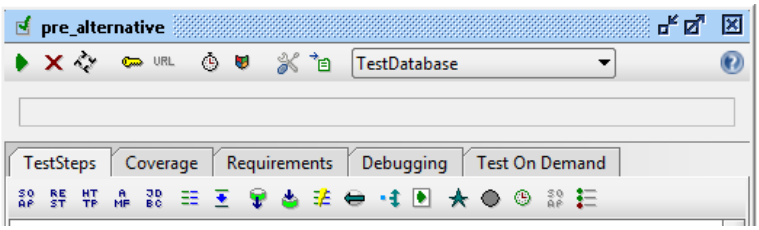


Obrázek 40: Nastavení proměnných pro projekt a testovací skupinu

Nyní bude popsána tvorba vybraného testovacího případu, který otestuje jeden z požadovaných průchodů a zkontroluje určené hodnoty. Nový testovací případ se vytvoří po vyvolání kontextové nabídky z testovací skupiny a zvolením volby `New TestCase`. Testovací kroky lze přidat vyvoláním kontextové nabídky z testovacího případu a zvolení volby `Add Step - požadovaný testovací krok`. Testovací krok lze zvolit i z nabídky v rámci testovacího případu. Nabídka je zobrazena na obr. 41.

V rámci vytvořeného testovacího případu byly využity následující typy testovacích kroků:

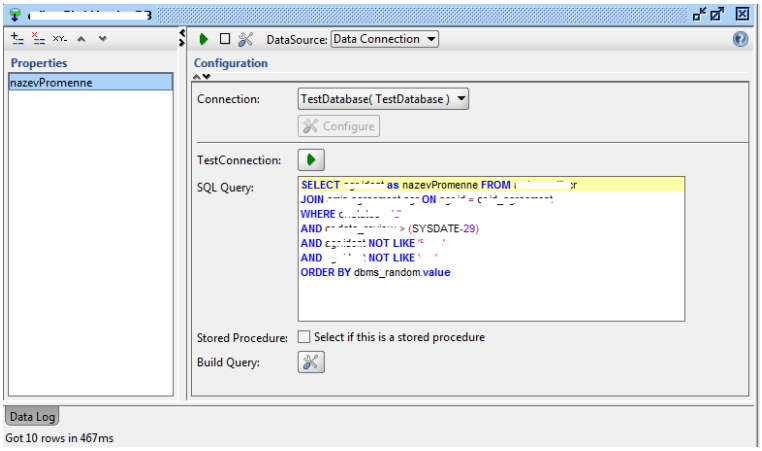
- `DataSource`,
- `GroovyScript`,
- `Assertion`,



Obrázek 41: Nabídka možností v rámci testovacího případu

- `Test Request`.

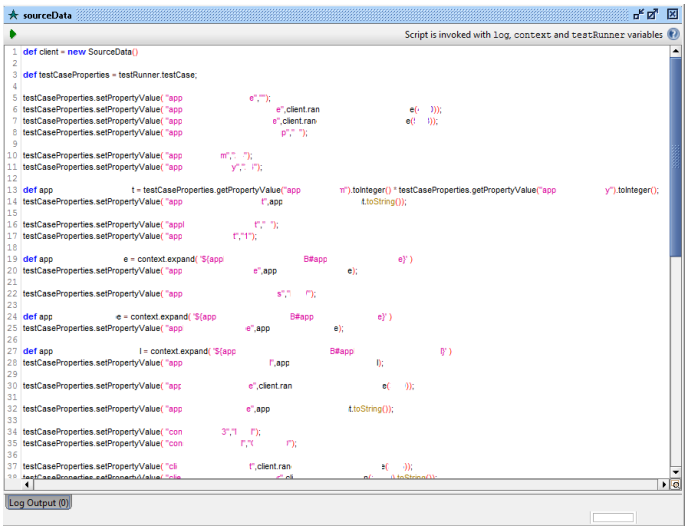
`DataSource` umožňuje mimo jiné možnost připojení na databázi. V nastavení je potřeba zvolit jako `DataSource` volbu `DataConnection`. Jelikož je již z předešlé části nastaveno konfigurační připojení na databázi, není potřeba jej znovu nastavovat. Stačí v sekci `Connection` vybrat požadované prostředí. Poté je možné vytvořit novou proměnnou a následně použít SQL dotaz, který do proměnné přiřadí výsledek dotazu. Aby se do proměnné nastavila požadovaná hodnota, je potřebné, aby se sloupec, ze kterého jsou získávány data, v databázi jmenoval stejně, jako název proměnné v `DataSource`. V opačném případě lze použít příkaz `AS` pro dočasné přejmenování názvu sloupce. Pokud je potřeba použít při sestavování dotazu data v rámci SoapUI, lze tak provést po vyvolání kontextové nabídky a volby `Get Data - výběr proměnné`. Ukázka použití je zobrazena na obr. 42



Obrázek 42: Použití `DataSource`

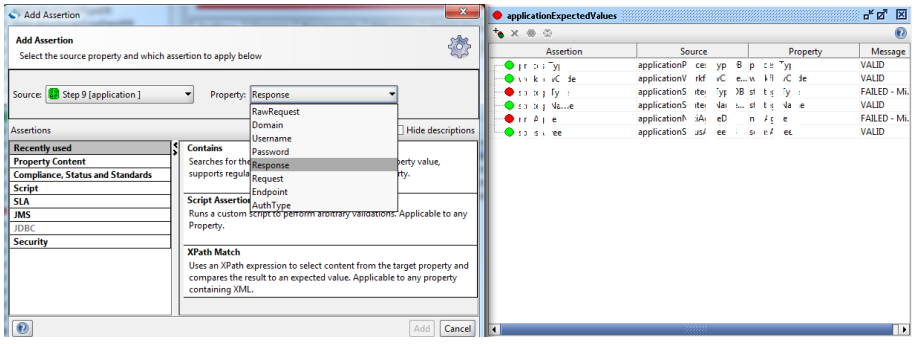
`GroovyScript` zajišťuje veškeré nastavení do proměnných v úrovni testovacího případu. Nastavení proměnných v úrovni testovacího případu je realizováno pomocí metody `setPropertyValue("nazevPromenne", "hodnota")`. Úroveň testovacího případu je přístupná přes příkaz `testRunner.testCase`. Pro získání proměnných je možné použít metodu `getPropertyValue("nazevPromenne", "hodnota")`. V rámci tohoto testovacího kroku lze vytvořit proměnnou, která představuje výsledek jiného testovacího kroku, například hodnotu z databáze a dále s ní pracovat. Po-

stup je stejný jako výše – vyvolání kontextové nabídky a volba **Get Data - výběr proměnné**. V tomto kroku jsou také volány metody z importovaného Groovy souboru. Na obr. 43 je zobrazen testovací krok.



Obrázek 43: Použití **GroovyScript**

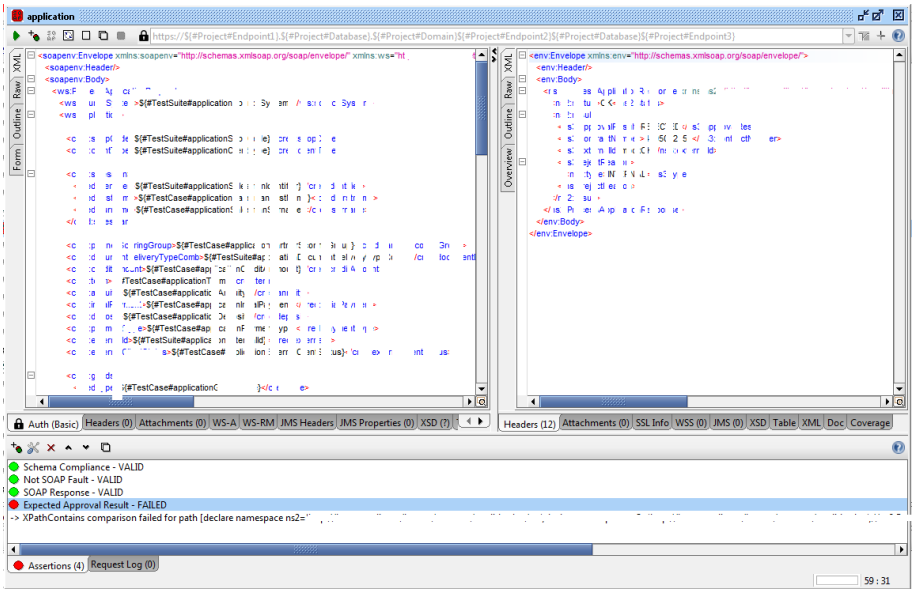
Assertion slouží pro kontrolu hodnot. Může být použit jako testovací krok i jako součást Test Request. Po vytvoření **Assertion** je možné zvolit, pro jakou proměnnou bude kontrola vykonána. Může to být například kontrola očekávané hodnoty v databázi, hodnoty v odpovědi (**Response**) na zaslaný požadavek v rámci Test Request apod. **Assertion** podporuje i kontrolu pomocí regulárních výrazů. Pokud jsou jednotlivé kontroly vyhodnoceny jako správné, jsou zobrazeny zelenou barvou. V opačném případě je prvek kontroly červený. Na obr. 44 je v levé části zobrazen výběr nové kontroly a v pravé části jsou zobrazeny již existující a vyhodnocené kontroly.



Obrázek 44: Použití **Assertion**

Test Request zajišťuje zasílání určených hodnot na adresu webové služby (**Endpoint**). Zasílaná data jsou přebírána z proměnných, které se nachází v úrovni testovací skupiny a aktuálního testovacího případu. Data v

rámci určitého prvku v testovacím požadavku jsou tedy deklarována pomocí – `${#TestSuiteNazevPromenne}` v případě proměnné z testovací skupiny a `${#TestCaseNazevPromenne}` v případě přebírání hodnoty z aktuálního testovacího případu. Po zaslání požadavku na adresu webové služby je obdržena odpověď a zároveň jsou provedeny určité akce v databázi a v neposlední řadě je obdržena odpověď. Ta obsahuje informace, na základě kterých se může provést zaslání navazujícího požadavku apod. Všechny údaje v odpovědi lze kontrolovat pomocí `Assertion`. Na obr. 45 je zobrazen testovací požadavek v levé části a odpověď v pravé části.



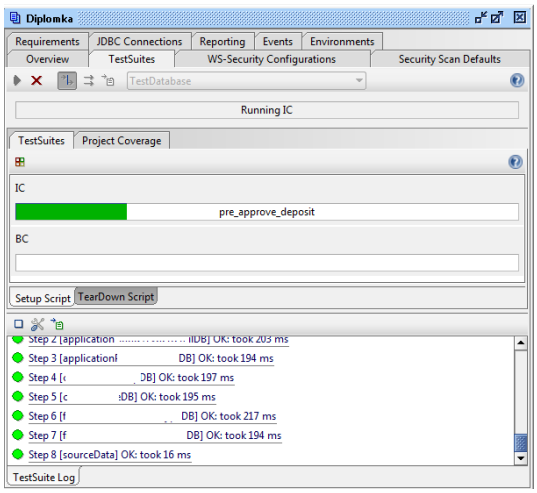
Obrázek 45: Použití `Test Request`

Spouštění testů

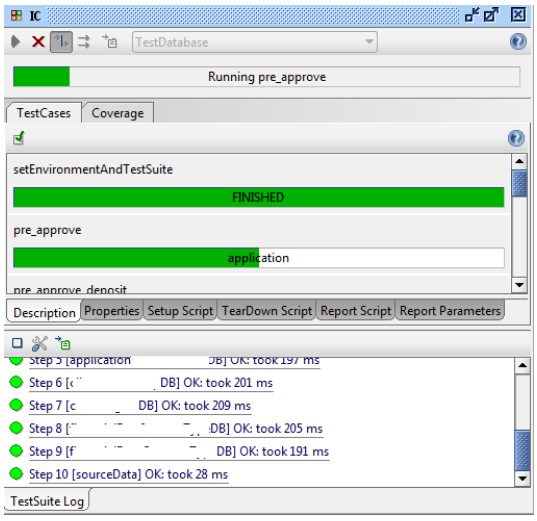
Spouštění testů je možné v následujících úrovních:

- **`Projekt`** – po výběru prostředí `TestDatabase` lze spustit všechny testovací skupiny v rámci projektu. V hlavní části je zobrazen právě probíhající testovací případ. V dolní části jsou v logu zobrazeny průběhy všech testovacích případů a testovacích kroků. Na obr. 46 je zobrazen průběh testů.
- **`Testovací skupina`** – po výběru prostředí prostředí lze spustit všechny testovací případy v rámci testovací skupiny. V hlavní části je zobrazen právě probíhající testovací případ a příslušný testovací případ. V logu v dolní části jsou poté opět všechny provedené kroky. Na obr. 47 je zobrazen průběh testů.
- **`Testovací případ`** – po výběru prostředí lze spustit všechny nebo jednotlivé testovací kroky v rámci testovacího případu. V hlavní části je seznam všech testovacích kroku a je zvýrazněn právě probíhající. Zobrazen je také v horní

části. V dolní části jsou v logu zaznamenány všechny provedené kroky. Na obr. 47 je zobrazen průběh testu.

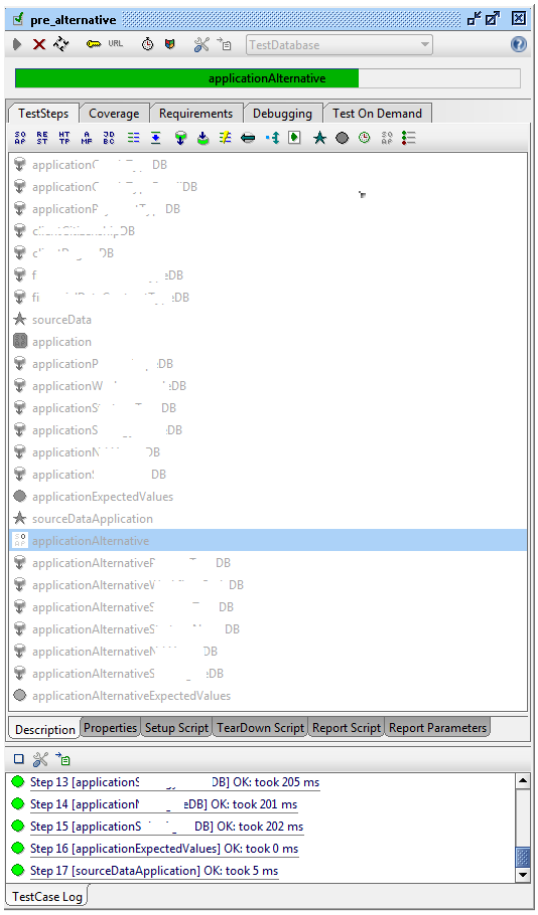


Obrázek 46: Probíhající testy v rámci projektu



Obrázek 47: Probíhající testy v rámci testovací skupiny

Testy lze spouštět i přes příkazovou řádku. Testy je možné spouštět v nastaveném počtu opakování. Po dokončení testů je možné vygenerovat zprávu o výsledku testování.



Obrázek 48: Probíhající test v rámci testovacího případu

4.5 Výsledný stav

Bylo vytvořeno několik desítek testovacích případů, z nichž každý obsahuje přibližně 15 - 30 testovacích kroků. Tyto testovací případy zahrnují požadované kombinace průběžných a konečných stavů. Sadu automatizovaných testů lze využít nejen při provádění průběžné kontroly funkcionality, ale také pro vytváření dat, která jsou prerekvizitou pro rozsáhlou oblast testování.

5 Závěr

V úvodní kapitole byla popsána problematika testování softwaru. Následně byly přiblíženy oblasti jako metodika vývoje softwaru, role v procesu testování, způsoby testování a především nástroje pro správu a řízení testů. Díky těmto nástrojům je proces testování efektivnější a pohodlnější.

V následující kapitole byly popsány výhody a nevýhody automatizovaného testování a přiblíženy techniky v procesu automatizovaného testování. Následovalo prozkoumání nástrojů určených pro vytváření automatizovaných a výkonnostních testů. U každého z těchto nástrojů byl proveden popis funkcionality a byla demonstrována praktická ukázka formou vytvoření jednoduchého testu. Zdrojové soubory z těchto příkladů jsou přiloženy na CD. Dále byly přiblíženy nástroje pro revizi a kontrolu kódu.

V poslední kapitole byly určeny požadavky na automatizaci. Na základě těchto požadavků byl vytvořen návrh testů. Poté byly vybrány nástroje, které byly použity pro vytvoření sady automatizovaných testů. Následně byla popsána metodika práce včetně detailních postupů a zhodnocení výsledného stavu.

Cílem práce bylo navrhnout a implementovat sadu automatizovaných testů pro testování funkčnosti bankovního softwaru, což se podařilo splnit. Sada testů je funkční a byla již mnohokrát použita v praxi. Mezi další přínosy práce lze zařadit přehled nástrojů včetně ukázkových příkladů, vytvoření návrhu a metodiky, pomocí které lze vytvářet automatizované testy.

Osobním přínosem bylo především seznámení se s nástroji, které se používají v oblasti testování. Dále to byla zkušenost s návrhem rozsáhlé sady automatizovaných testů, následné implementace a získané vědomosti a poznatky z těchto procesů.

V rámci inovace sady automatizovaných testů je možné tuto sadu provázat s nástrojem Jenkins. Ten umožňuje vytvořit opakované spouštění procesů. Bylo by tak možné nastavit spouštění kompletní sady testů každý den. Po skončení testů by se na předem určené osoby automaticky zaslal email ohledně výsledků testů. Do budoucna je plánována průběžná implementace dalších automatizovaných testů.

6 Reference

ATLASSIAN *Crucible.*[online]. [cit. 2015-05-01]. Dostupné z: `https://www.atlassian.com/software/crucible`.

BINARYCODER *Using FxCop: A Short Tutorial.*[online]. [cit. 2015-04-28]. Dostupné z: `http://www.binarycoder.net/fxcop/html/tutorial.html`.

BRUCKNER, T. *Tvorba informačních systémů: principy, metodiky, architektury.* 1. vyd. Praha: Grada, 2012, 357 s. Management v informační společnosti. ISBN 978-80-247-4153-6.

DEMARCO, T. *Peopleware: productive projects and teams.* 2nd ed. New York: Dorset House Publishing, 1999, 245 s. ISBN 978-0-932633-43-9.

DIJSKTRA, E., W. *ACM Turing Award Lectures – The humble programmer.* New York: Association of Computing Machinery, 1972. Dostupné z: `http://dl.acm.org/citation.cfm?id=1283927.` DOI: 10.1145/1283920.1283927.

EELES, P., CRIPPS P. *Architektura softwaru.* Vyd. 1. Brno: Computer Press, 2011, 328 s. ISBN 978-0321357489.

HETZEL, W., C. *Program test methods.* Englewood Cliffs, N.J.: Prentice-Hall, 1973, 311 s. ISBN 01-372-9624-X.

HYNAR, M. *Java: nástroje.* Praha: Neocortex, 2004, 325 s. ISBN 80-863-3016-8.

IEEE *IEEE standard for software and system test documentation.* [online]. New York, NY: Institute for Electrical and Electronics Engineers, 2008 [cit. 2015-04-01]. Dostupné z: `http://www.computing.dcu.ie/~davids/courses/CA267/ieee829mtp.pdf.` ISBN 9780738157467.

KENNET, R., BAKER, E. *Software process quality: management and control.* New York: Marcel Dekker, 1999, 241 s. Computer aided engineering (New York, N.Y.), 6. ISBN 08-247-1733-3.

LEWIS, W., DOBBS, D., VEERAPILLAI, G. *Software testing and continuous quality improvement.* 3rd ed. Boca Raton: CRC Press, 2009, 655 s. ISBN 14-200-8073-3.

MICROSOFT ACE TEAM *Performance testing Microsoft .NET Web applications.* Redmond, Wash.: Microsoft Press, 2003, 284 s. ISBN 07-356-1538-1.

MORGAN, P., SAMAROO A., HAMBLING B. *Software testing: an ISEB foundation.* 2nd ed. London: British Computer Society, 2010. ISBN 978-190-6124-762.

Myers, G., J. *The art of software testing.* 3rd ed. Hoboken, New Jersey: Wiley, 2012, 240 s. ISBN 978-1-118-03196-4.

Page, A., Johnston, K., Rollison, B. *How we test software at Microsoft.* Redmond, WA: Microsoft Press, 405 s. ISBN 978-073-5624-252.

Patton, R. *Software testing.* 2nd ed. Indianapolis: Sams Publishing, 2006, 389 s. ISBN 06-723-2798-8.

PractiTest *Product.*[online]. [cit. 2015-04-01]. Dostupné z: `http://www.practitest.com/product/test-management`.

Roudenský, P., Havlíčková, A. *Řízení kvality softwaru: Průvodce testováním.* 1. vyd. Brno: Computer Press, 2013, 208 s. ISBN 978-80-251-3816-8.

SmartBear *Manage All Your Tests Centrally.* [online]. 2015 [cit. 2015-04-03]. Dostupné z: `http://smartbear.com/product/test-management-tool`.

Testhouse *HP Quality Center.* [online]. 2015 [cit. 2015-04-06]. Dostupné z: `http://www.testhouse.net/hp`.