Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informačních technologií

Porovnání nástrojů pro automatizaci testů v rámci vývoje SW

Bakalářská práce

Autor: Jan Preisler
Studijní obor: Aplikovaná informatika

Vedoucí práce: doc. Ing. Vladimír Bureš, Ph.D., MBA

Odborný konzultant: Ing. Miroslav Ježek
Unicorn Systems a.s.

Hradec Králové duben 2019

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 19.4.2019 Jan Preisler

Poděkování:

Děkuji vedoucímu bakalářské práce doc. Ing. Vladimíru Burešovi, Ph.D., MBA za metodické vedení práce a Ing. Miroslavu Ježkovi ze společnosti Unicorn Systems a.s. za odborný dohled a užitečné rady.

Anotace

Cílem bakalářské práce je seznámit čtenáře s problematikou testování softwaru a představit proces automatizace testů.

Práce je rozdělena na dvě části. První část je věnována teorii testování. V rámci této části jsou představeny metodiky testování, druhy testů, ale také životní cyklus testování nebo druhy chyb, které vznikají při vývoji softwaru. Jsou zde také zmíněné důvody, proč vůbec chyby vznikají. Závěr první části se zabývá automatizovanému testování, kde jsou uvedeny důvody, proč testy automatizovat společně s výhodami a nevýhodami automatizace. Dále také na proces automatizace a nejčastější kritéria, která jsou důležitá při volbě nástroje pro automatizace testů.

V druhé části jsou představeny 3 vybrané nástroje, které jsou následně porovnány na základě zvolených kritérií.

Annotation

Title: Comparison of tools for automation of tests in SW development

The aim of the bachelor thesis is to acquaint the reader with the issue of software testing and introduce the process of automatization of tests.

The thesis is divided into two parts. The first part is dedicated to testing theory. This part introduces testing methodologies, types of tests, and the testing life cycle or types of errors that occur in development of software. Here there are mentioned also the reasons why these errors arise. The conclusion of the first part is dedicated to automated testing, where are the reasons, why tests automate together with the advantages and disadvantages of automation. Next, also the automation process and the most common criteria that are important in choosing a tool for automation.

In the second part there is introduction of 3 selected tools, which are then compared based on selected criteria.

Obsah

1 Úvod ... 5

2 Cíle a metodiky práce ... 6

3 Testování ... 7

3.1 Životní cyklus testování ... 7

3.1.1 Analýza požadavků ... 7

3.1.2 Plánování testu ... 7

3.1.3 Návrh testů ... 8

3.1.4 Nastavení testovacího prostředí ... 8

3.1.5 Provedení testu ... 8

3.1.6 Uzavření testovacího cyklu ... 9

3.2 Chyba ... 9

3.2.1 Proč chyby vznikají ... 9

3.2.2 Kdy je vhodné testovat a náklady ... 10

3.2.3 Druhy chyb ... 10

3.3 Metody testování ... 12

3.3.1 Statické testování ... 12

3.3.2 Dynamické testování ... 12

3.4 Druhy testů ... 14

3.4.1 Jednotkový test (Unit test) ... 14

3.4.2 Integrační test ... 15

3.4.3 Smoke testy ... 16

3.4.4 Regresní testy ... 17

3.4.5 Systémový test ... 17

3.4.6 Akceptační testování ... 17

4 Automatické testování ... 18

4.1 Výhody a nevýhody automatizace testů 18
4.2 Proces automatizace testů 19
4.2.1 Výběr nástroje 19
4.2.2 Definice rozsahu automatizace 19
4.2.3 Plánování, návrh a vývoj 20
4.2.4 Vykonání testu 20
4.2.5 Údržba 20
4.3 Nástroje pro automatické testování 21
4.3.1 Kritéria pro výběr nástroje 21
5 JMeter 23
5.1 Instalace 23
5.2 Uživatelské rozhraní 23
5.3 Práce s nástrojem 25
5.3.1 Implementace testu 25
5.3.2 Úprava testovacího skriptu 26
5.3.3 Spuštění testu 27
5.3.4 Analýza výsledků 27
5.4 Zhodnocení nástroje 27
6 IBM Rational Functional Tester (RFT) 28
6.1 Instalace 28
6.2 Uživatelské rozhraní 28
6.3 Práce s nástrojem 29
6.3.1 Implementace testu 30
6.3.2 Verifikační body 30
6.3.3 Parametry v testovém skriptu 31
6.3.4 Úprava testovacího skriptu 31

6.3.5 Spuštění testu 31
6.3.6 Analýza výsledků 32
6.4 Zhodnocení nástroje 33
7 Sahi 34
7.1 Instalace 34
7.2 Uživatelské rozhraní 34
7.3 Práce s nástrojem 35
7.3.1 Implementace testu 36
7.3.2 Verifikační body 36
7.3.3 Úprava testovacího skriptu 36
7.3.4 Spuštění testu 37
7.3.5 Analýza výsledků 37
7.4 Zhodnocení nástroje 38
8 Porovnání nástrojů 39
8.1 Použitelnost nástroje 39
8.2 Křivka učení 40
8.3 Správa a znovupoužitelnost testových skriptů 41
8.4 Integrace na jiné nástroje 41
8.5 Analýza výsledků 42
8.6 Konečné hodnocení 42
9 Závěr 43
10 Seznam obrázků 44
11 Seznam tabulek 45
Seznam použité literatury 46

1 Úvod

S rostoucí složitostí jednotlivých softwarových řešení napříč všemi odvětvími rostou také nároky na kvalitu. Není vhodné a někdy dokonce ani přípustné, aby se v produktu, po předání zákazníkovi, vyskytovaly chyby, které mohou mít zásadní vliv na chod softwaru. Na základě toho se ve vývojovém cyklu stále častěji začalo objevovat testování. Testování také umožňuje bezproblémové rozšiřování a údržbu softwaru, neboť při nějaké změně nebo přidání funkcionality stačí spustit testy a je hned zřejmé, zda tato úprava neměla za následek nefunkčnost jiné části softwaru. Testování softwaru si tak vydobylo svoje místo ve vývoji. Dnes už je téměř každému jasné, že je důležité této problematice věnovat určitou pozornost, a tak se tomu dnes věnují celé týmy a z testování se stal samostatný obor.

V dnešní rychlé a dynamické době se neustále objevují nové technologie a postupy a s tím přichází i nové druhy chyb, na které je potřeba během vytváření softwaru brát ohled. Na základě toho je potřeba, aby se i testování neustále vyvíjelo a bylo schopné odhalovat tyto chyby.

Protože existuje celá řada testů, ať už na testování funkcionality, nebo na testování výkonnosti, a také neustále nové druhy testů vznikají, není téměř možné, aby veškeré testy prováděli testeři manuálně. Tester nemá čas vracet se ke starým testům a provádět je znovu. Objevila se tedy potřeba testy automatizovat. Automatizace zrychluje vývoj a zajišťuje požadovanou kvalitu.

Díky tomu začali vznikat nástroje, které tuto automatizaci umožňují. V dnešní době existuje na trhu velké množství různých nástrojů nebo frameworků, ať už bezplatných, nebo komerčních. Nástroje jsou určené na různé testy a téměř žádný nástroj neumí provádět veškeré typy testů. Pro výběr nástroje tedy existuje spoustu kritérií, které je nutné při výběru zohlednit. Nástroje se vybírají přímo pro daný tým testerů, kteří s ním budou pracovat.

2 Cíle a metodiky práce

Prvním cílem práce je seznámit čtenáře s testováním softwaru včetně základních pojmů a postupů, které se při testování používají. Druhým cílem je porovnání třech vybraných nástrojů na základě několika kritérií.

Práce je rozdělena na praktickou a teoretickou část. Na začátku teoretické části jsou představeny jednotlivé kroky, které tvoří životní cyklus testování. Následuje podkapitola, která je věnována chybám, kde je jsou popsány nejčastější příčiny vzniku chyb a druhy chyb, které mohou při vývoji vzniknout. Na to navazují metody testování a jednotlivé druhy testů. Nemalá část je věnována automatizaci testů, kde jsou uvedeny důvody, proč je vhodné testy automatizovat včetně výhod a nevýhod. V této kapitole je také představen proces automatizace a nejčastější kritéria, která hrají roli při výběru nástroje.

V praktické části jsou představeny tři nástroje určené pro automatizaci testů, kde u každého nástroje jsou popsány druhy testů, pro který je nástroj primárně určen. Do popisu nástroje je zahrnut postup při instalaci, představení uživatelského rozhraní, a také samotná práce s nástrojem.

Tyto nástroje byly vybrány podle oblíbenosti mezi testery, ceny, ale také dostupnosti dokumentace.

Nástroje byly porovnány na základě určených kritérií. Podklady pro toto porovnání vyplynuly z implementace testovacího scénáře do každého nástroje. Jednalo se o test uživatelského rozhraní. Test byl založen na vyhledání určitého filmu v Česko-Slovenské filmové databázi. Testování bylo prováděno na počítači s operačním systémem Windows 10 edice Home Premium. Pro testování nástrojů IBM Rational Functional Tester a Sahi Pro byly využity třicetidenní zkušební licence.

3 Testování

Doba jde dopředu a výpočetní technika se dnes používá téměř v každém odvětví. Je tedy logické, že se složitost softwaru a nároky na jeho kvalitu stále zvyšují. I z tohoto důvodu se stalo testování nedílnou součástí vývojového cyklu softwaru. Testování je tedy proces určení správnosti a kvality softwarového programu. Hlavním úkolem testování je projít daný software za účelem nalezení většiny chyb, tyto chyby zaznamenat a zajistit jejich nápravu [1]. Bohužel, ani za pomocí testování není možné nalézt a opravit veškeré chyby. Zvlášť u komplexního softwaru, kde je množství vstupních a výstupních kritérií téměř nekonečné [2].

3.1 Životní cyklus testování

Životní cyklus testování je proces zajišťující systematické a plánované provádění operací vedoucích ke zlepšení kvality produktu. Každý krok procesu má svá vstupní a výstupní kritéria. Je tedy žádoucí, aby každý další krok byl založen na kroku předchozím, v praxi však toto není vždy pravda [3].

3.1.1 Analýza požadavků

Analýza požadavků je prvním z šesti kroků životního cyklu testování. Během tohoto kroku se zkoumají veškeré požadavky a vybírají se požadavky ty, které budou dále testované. V případě jakékoliv nejasnosti komunikuje tým testerů s ostatními zainteresovanými stranami (softwarový architekt, technický manažer, klient…). Na základě této analýzy je vytvořen RTM (Requirements Traceability Matrix) dokument, který mívá nejčastěji formát tabulky a zahrnuje veškeré testovatelné požadavky. V této fázi se také zkoumá proveditelnost automatizace [3][4].

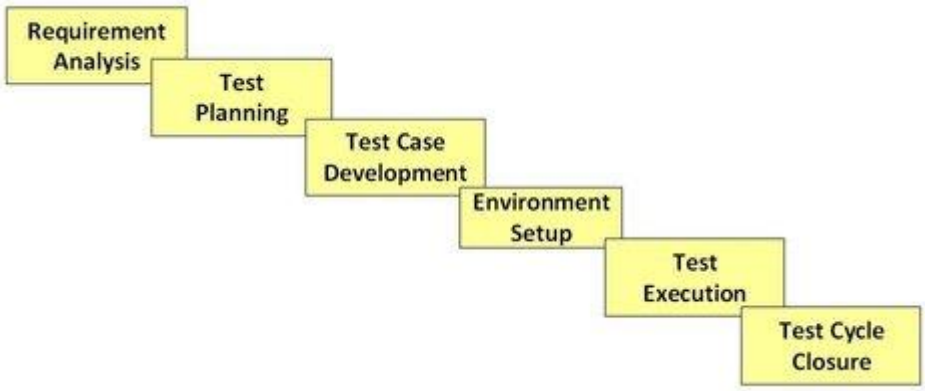
3.1.2 Plánování testu

Dalším krokem je plánování testu. Tato fáze je nejdůležitější v celém životním cyklu testování. Hlavními úkoly je určit testovací strategii, odhadnout časovou náročnost a v neposlední řadě také určit odhad celkových nákladů. Plánování testu je zahájeno poté, co je dokončena analýza požadavků. Na základě analýzy se začínají

připravovat testovací plány. Výsledkem této fáze je testovací strategie a dokument popisující časovou náročnost a odhad celkových nákladů [3].

3.1.3 Návrh testů

Po dokončení plánovací fáze se začínají navrhovat a vytvářet jednotlivé testy, které ověří, zda jsou všechny požadavky na software splněny. Pokud je k dispozici testovací prostředí, je nutné vedle jednotlivých testů připravit také testovací data [3].



Obrázek 1: Životní cyklus testování
Zdroj: [4]

3.1.4 Nastavení testovacího prostředí

Velmi důležitou fází životního cyklu testování je nastavení testovacího prostředí. Nastavení testovacího prostředí udává, za jakých podmínek bude software testován.

Pokud testovací prostředí poskytuje vývojářský tým nebo zákazník, musí testovací tým vytvořit tzv. smoke testy, které ověří připravenost prostředí. O těch ale později [3][4].

3.1.5 Provedení testu

Jakmile je dokončeno testování prostředí, může být zahájeno samotné testování softwaru. Tým testerů provádí jednotlivé testy podle připraveného scénáře. Pokud se během testu vyskytne chyba, je tato chyba předána vývojářům,

kteří zajistí její opravu. Když je chyba opravena, může být proveden stejný test znovu [3].

3.1.6 Uzavření testovacího cyklu

V poslední fázi probíhá diskuze mezi testery, kde se hodnotí průběh testování a zkoumají se problémy, které nastaly během testů. Analýza těchto problémů slouží jako vstupní kritérium do dalších testů a pomáhá zlepšovat kvalitu testování.
Na závěr testovacího cyklu probíhá analýza výsledku testu. Výsledkem je rozdělení jednotlivých závad podle typu a závažnosti [3].

3.2 Chyba

Softwarová chyba, nebo jednoduše anglicky bug, je vada v počítačovém programu, která má za následek neočekávané chování nebo neočekávané hodnoty výstupů. Chyby mohou být jemného rázu nebo díky výskytu závažné chyby může běh programu způsobit selhání celého operačního systému [5].

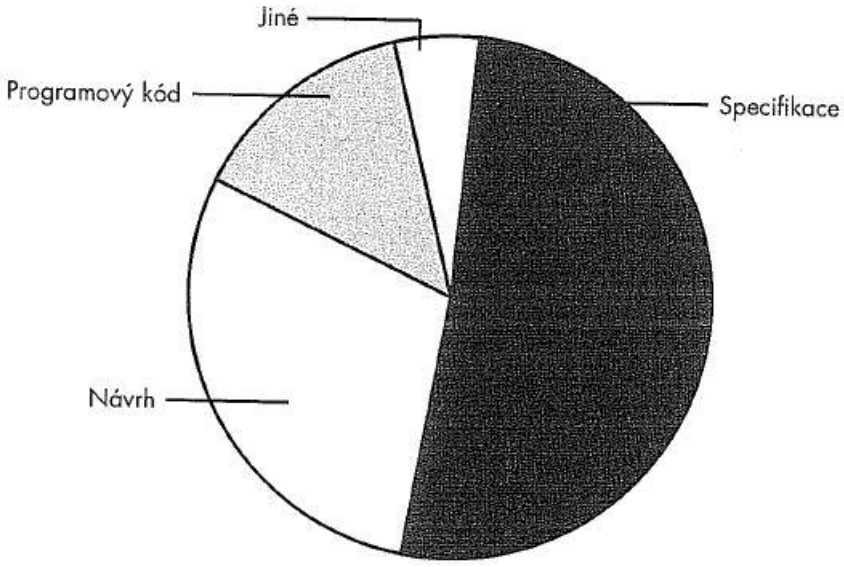
3.2.1 Proč chyby vznikají

Člověk nezasvěcený do problematiky testování by řekl, že chyby nejčastěji vznikají chybným zápisem ve zdrojovém kódu. Tato domněnka je ovšem mylná. Nejčastější příčinou bývá už specifikace. Existuje několik důvodů, proč právě specifikace bývá nejčastějším zdrojem chyb. Pokud se plánování softwaru nedává dostatečná důležitost, může se stát, že specifikace úplně chybí nebo není dostatečně podrobná. Dalším důvodem bývá často měnící se specifikace nebo nedostatečná komunikace ve vývojovém týmu [6].

Mezi neopominutelné příčiny patří například chybný návrh softwaru, v němž chyby vznikají ze stejných důvodů, jako je tomu u specifikace. Pokud se neprovede instalace nebo konfigurace testovacího prostředí správně, může to také vést ke vzniku chyb nebo k chybnému provedení testu [6].

Až na třetím místě jsou poruchy způsobené chybným zdrojovým kódem. Ovšem většina těchto chyb vyplývá právě z chybné specifikace nebo z chybného

návrhu. Mezi další časté příčiny výskytu problému bývá časová tíseň nebo neustálé aktualizace a opravy kódu [6].



Obrázek 2: Nejčastější příčiny chyb
Zdroj: [6]

3.2.2 Kdy je vhodné testovat a náklady

Software vzniká systematickým postupem na základě vývojového modelu. Chyby je tedy možné opravovat během celého vývojového cyklu, avšak s rostoucím stadiem vývoje rostou i náklady na opravu chyb [6]. Pokud je chyba objevena hned na začátku vývoje, je možné provést nápravu za zlomek ceny, než by náprava stála, pokud by se chyba vyskytla až na konci celkového vývoje nebo dokonce po předání produktu zákazníkovi. Na druhou stranu ale není výhodné provádět testování hned ze začátku vývoje, jelikož testů by bylo prováděno příliš málo. Samozřejmě není ani vhodné testovat až na úplném konci, zde by testů bylo naopak příliš mnoho. Optimální řešení je testovat průběžně, a vždy v nějaké optimální době.

3.2.3 Druhy chyb

Existují chyby několika typů. Je důležité umět rozeznat jakého typu daná chyba je. Porozumění napomáhá k rychlé a správné opravě chyby [7].

3.2.3.1 Chyby funkcionality

Funkcionalita udává, jakým způsobem se má software chovat. Pokud po provedení akce nastane neočekávaná reakce, je toto chování dosti zmatečné a v některých případech může tato chyba znemožnit ovládání celého softwaru.

Příkladem je tlačítko uložit, které má za úkol uložit záznam. Pokud po stisknutí tohoto tlačítka nedojde k žádné akci, jedná se o chybu funkcionality [7].

3.2.3.2 Chyby komunikace

Chyby komunikace jsou například chybějící informace, které by uživatel měl vědět při používání softwaru. Pokud například uživatel provádí objednávku na e-shopu a po kliknutí na tlačítko odeslat se nepovede objednávka odeslat, měl by o tom uživatel být informován [7].

3.2.3.3 Chyby chybějících prvků

Tato chyba nastane, pokud uživatel nemá k dispozici veškeré ovládací prvky, které potřebuje. Touto chybou se rozumí například, pokud uživatel nemá možnost zavřít okno, které sám otevřel [7].

3.2.3.4 Syntaktická chyba

Syntaktická chyba vzniká chybným zápisem kódu a tím porušením gramatiky daného programovacího jazyka. Tyto chyby je jednoduché nalézt a opravit. Většina vývojových prostředí upozorňuje vývojáře na chybu, a také navrhuje její opravu.

3.2.3.5 Chyba při manipulaci s chybami

Veškeré informace o chybách, které vznikají při interakci s uživatelem, musí být dostatečně podrobné. Pokud během interakce s uživatelem nastane nějaká chyba a uživatel o ní není dostatečně informován nebo není informován vůbec, jedná se o chybu při manipulaci s chybami [7].

3.2.3.6 Výpočetní chyby

Výpočetní chyba může vzniknout použitím špatné logiky nebo špatného matematického vzorce. Velmi častou příčinou bývá nedodržení priorit početních

operací nebo neshodné datové typy [7]. V komplexním softwaru, kde existuje několik set tisíc výpočtů, které jsou na sobě závislé, nebývá jednoduché takovou chybu nalézt.

3.2.3.7 Chyba kontroly toku

Řídící tok udává, jak se bude software chovat po splnění určitých podmínek. Pokud po provedení určitých akcí a splnění daných podmínek nenastane očekávaná reakce, jedná se o chybu kontroly toku [7].

3.3 Metody testování

3.3.1 Statické testování

V rámci této techniky se provádí testování bez spuštění kódu. S daným softwarem se tedy nepracuje, ale pouze se prochází a kontroluje. Vzhledem k této skutečnosti je možné provádět testy již v rané fázi vývoje, kdy ještě není k dispozici funkční prototyp aplikace. Statické testování by nemělo nahrazovat dynamické testování, mělo by být ale jednou z hlavních částí testovacího cyklu. Hodí se především na kontrolu specifikace požadavků nebo na analýzu kódu [8]. Statické testování je prvním krokem kontroly kvality softwaru. Výhodou této metody je možnost odhalit vážné chyby a problémy ještě předtím, než se skutečně projeví. Toto testování ovšem neprovádí testeři, ale provádí ho samotní vývojáři. V dnešní době se na statické testování používají nástroje, které tyto testy zcela automatizují anebo je značně ulehčují.

3.3.2 Dynamické testování

Jak už název napovídá, dynamické testování se provádí na spuštěném softwaru. Testování se provádí až v pozdější fázi, jelikož je potřeba spustitelná verze kódu. V průběhu testování se sledují jednotlivé výstupy a porovnávají se s očekávanými výstupy. Mimo sledování výstupů se testování zaměřuje i na sledování systémové paměti, doby odezvy procesoru a celkového výkonu systému. Mezi hlavní techniky dynamického testování patří testování černé skříňky, testování bílé skříňky nebo testování šedé skříňky [8].

3.3.2.1 Testování černé skříňky

Testování černé skříňky je princip testování, při kterém tester nemá přístup ke zdrojovému kódu. Neví tedy jakým způsobem software pracuje. Ví pouze co by měl dělat. Při testování tester zadává vstupy a sleduje a vyhodnocuje výstupy, neví ale jak k danému výstupu software dospěl [6].

Techniky:

- **Rozdělení na třídy ekvivalence** – testovací podmínky jsou rozdělené do jednotlivých tříd, od kterých se očekává, že se budou chovat stejným způsobem. Poté se testuje jedna podmínka ze skupiny. Pokud funguje tato podmínka, musí fungovat i ostatní podmínky [9].
- **Analýza hraniční hodnoty** – testují se hranice hodnot, které jsou povoleny, nemusí se testovat všechny hodnoty, ale pouze hodnoty nad a pod hranicí [9].
- **Testování rozhodovacích tabulek** – v sloupcích tabulky jsou zapsané jednotlivé podmínky, ve sloupcích pak pravidla, která mají formu ano\ne. Kombinace řádků a sloupců určuje chování aplikace a představuje vstupy pro jednotlivé akce.

Tabulka 1: Výhody a nevýhody testování černé skříňky

Výhody	Nevýhody
• Tester nemusí znát programovací jazyky • Testovací případy mohou být navrženy po dokončení specifikací • Testy se provádí z pohledu uživatele	• Testování malého množství vstupů • Složitost navržení testovacích případů • Nemožnost cílit na části kódu náchylné k chybám

Zdroj dat: [9]

3.3.2.2 Testování bílé skříňky

Při testování bílé skříňky má tester, na rozdíl od černé skříňky, přístup ke zdrojovému kódu. Z možné analýzy může tester určit, které hodnoty povedou více či méně k chybám a podle toho upravit další testování [6].

„Testování bílé skříňky s sebou přináší jedno riziko. Člověk při něm velice snadno sklouzne k tomu, že testování „ušije na míru" činnosti programového kódu a nepodaří se mu software otestovat opravdu objektivně." [6]

Tabulka 2: Výhody a nevýhody testování bílé skříňky

Výhody	Nevýhody
• Pro testování se nemusí čekat až na implementaci GUI • Pomáhá při optimalizaci kódu • Důkladnější testování	• Nutná důkladná znalost programování • Údržba testovacích skriptů.

Zdroj dat: [6]

3.3.2.3 Testování šedé skříňky

Testování šedé skříňky je kombinace předešlých principů. Tester má tedy částečný přehled o fungování softwaru. Ve většině případů tester zkouší daný software přes uživatelské rozhraní. Výstupy si poté může ověřit například pomocí dotazů do databáze [10].

3.4 Druhy testů

3.4.1 Jednotkový test (Unit test)

Jednotkový test probíhá na nejnižší úrovni, jelikož testuje komponentu odděleně od ostatních komponent. Pokud komponenta závisí na jiných komponentách, jsou tyto komponenty nahrazeny pomocí takzvaných stubs nebo mocks. Díky tomuto nahrazení je možné testovat komponentu samostatně a nemusí se řešit chyby ostatních komponent. Stub nebo mock simuluje správné chování ostatních komponent. Tato substituce přináší možnost nahradit např. databázi třídou, která bude databázi simulovat. Odstíníme tedy dobu odezvy serveru a odpovědi nebo příkazy na server proběhnou okamžitě [11].

Je vhodné se jednotkovými testy zabývat už ve fází návrhu softwaru a rozhodnout, zda se budou tyto testy používat. Aplikace takovýchto testů na zaběhlých projektech bývá složitá a vyžaduje četné úpravy v kódu [11].

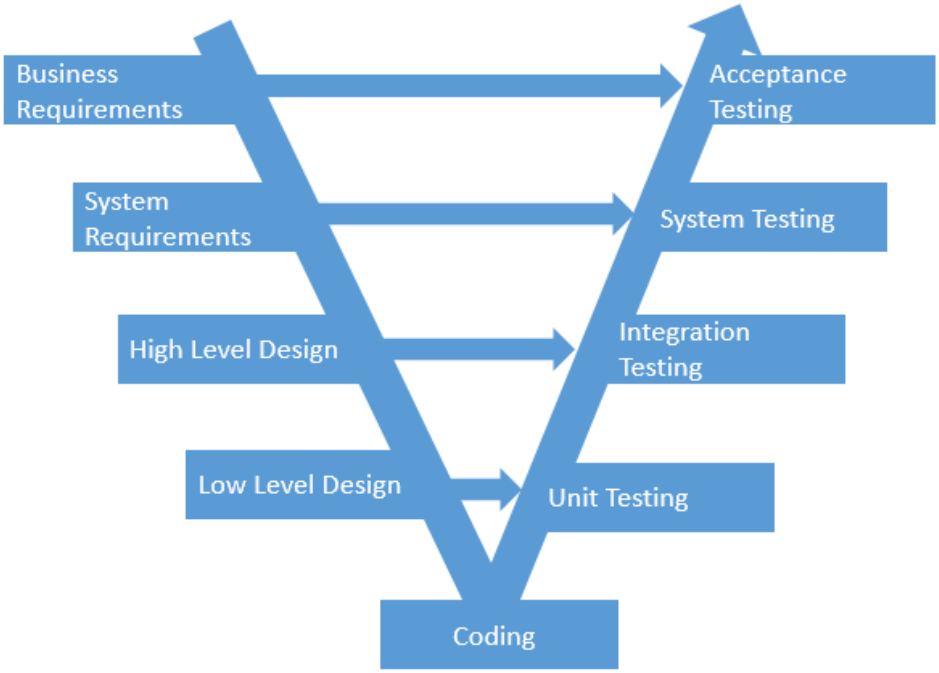
Jednotkové testy si v drtivé většině píší vývojáři sami. Používaní těchto testů značně ulehčuje údržbu. Velkou výhodou je, že při změně nějaké funkcionality nebo nějaké části kódu se chyba objeví už během průběžné integrace, a ne až v produkci. Průběžná integrace znamená, že každý vývojář integruje část svojí práce pravidelně. V optimálním případě denně. Každá integrace se ověřuje automatickými testy a umožňuje rychlé nalezení chyb.

3.4.2 Integrační test

Integrační test je někdy také označovaný jako test vnitřní integrace. Ověřuje komunikaci mezi jednotlivými komponentami, mezi komponentou a operačním systémem/hardwarem nebo mezi komponentou a rozhraním jiného systému. Test začíná mezi dvěma komponentami a poté se do testu přidávají další. U menších testů bývá obvyklé integrační test vynechat. Chyba, která by se objevila při integračním testu se jistě projeví i v další fázi testování. Pokud se ovšem chyba projeví už v integračním testu, jsou náklady a úsilí nutné k nápravě chyby značně menší, než kdyby se chyba projevila až později. Integrační testy tedy mají svůj význam [11].

Proč rozlišovat jednotkové a integrační testy [12]:

- **Rychlost** – díky testování komponenty odděleně a použití mocku je jednotkové testování značně rychlejší
- **Simulace chybových stavů** – díky použití stubu nebo mocku je možné simulovat různé chybové stavy
- **Nezávislý vývoj** – možnost vyvíjet nezávisle dvě závislé komponenty



Obrázek 3: V model testovacího procesu
Zdroj: [13]

3.4.3 Smoke testy

Smoke testy ověřují, zda lze daný software spustit a lze na něm provádět další testy nebo s ním pracovat. K provedení této kontroly je potřeba spustitelná verze testovaného produktu, a proto se provádí na konci fáze integračních testů.

Hlavním cílem těchto testů je ověřit, jestli jsou implementovány a spuštěny všechny potřebné části softwaru. Kontrolují se pouze hlavní funkce, které se často nemění. Díky tomu a díky rozsahu jsou většinou tyto testy automatizované, a to buď úplně nebo alespoň částečně [11].

Úspěšné dokončení této kontroly je podmínkou pro vstup do další fáze testování. Nemá totiž smysl testovat produkt, který není možné spustit nebo u kterého nefungují jeho hlavní funkce.

3.4.4 Regresní testy

Je nutné ověřovat, zda software, který byl už dříve testován funguje stejným způsobem i poté, co je přidána nová funkcionalita, opravena chyba, změněna konfigurace nebo potom, kdy je software napojen na jiný systém. Regresní testy je vhodné automatizovat. Během testování regrese se mohou objevit nové chyby [14]. Regresní testy by se měly provádět často, neboť každá změna v softwaru může vést ke vzniku nových chyb.

3.4.5 Systémový test

Systémové testy se provádí v pozdější fázi vývoje, protože testování se provádí z pohledu zákazníka a software se testuje jako funkční celek. Je to tedy poslední fáze testování před předáním produktu zákazníkovi. Testy probíhají podle předem připravených scénářů. Vzhledem k tomu, že systémový test je jakousi výstupní kontrolou, je vhodné věnovat těmto testům velkou pozornost při návrhu postupu testování, aby testovací scénáře byly co nejvíce přizpůsobené softwaru [11].

V rámci tohoto testu se ověřuje funkčnost softwaru pro různé operační systémy. Využívá se princip černé skřínky. Testera tedy nezajímá, co se děje uvnitř softwaru, pouze zadává požadované vstupy a sleduje výstupy. Součástí systémového testu bývá například testování bezpečnosti, výkonu nebo testování zotavení [15].

3.4.6 Akceptační testování

Akceptační testy jsou poslední fází testování. Testy si provádí zákazník na svém testovacím prostředí podle scénářů, jež byly připraveny ve spolupráci s dodavatelem. Nalezená chyba je předána zpět vývojovému týmu, který zajistí její opravu v co nejkratší době. Opravené chyby se opět testují v prostředí u zákazníka [11].

4 Automatické testování

Testování je náročnou a nedílnou součástí vývojového cyklu softwaru. Potřebuje tedy svůj čas. Provádění všech testů manuálně je neefektivní a vzniká zde riziko, že test bude proveden chybně. Rozdělení testovacích případů do tříd ekvivalence a poté testovat vždy jeden případ ze skupiny sice šetří čas, ale je zde možnost, že z testování bude vynechána nějaká důležitá funkce [6]. Vhodným způsobem, jak ušetřit čas, a tudíž i finance, je zautomatizování některých testů.

Základním kamenem je použití vhodných nástrojů, které v jisté míře zjednoduší provádění testů. Existuje řada nástrojů, které umožňují nejen spouštění připravených testů, ale i automatizaci části nebo celého procesu testování.**Chyba! Nenalezen zdroj odkazů.**

4.1 Výhody a nevýhody automatizace testů

Nejhlavnější výhodou automatizace testů je úspora času. Testy je možné spouštět kdykoliv, je tedy vhodné nechat nástroje testovat přes noc, kdy s daným softwarem nepotřebují pracovat jiné týmy, například týmy vývojové. Mezi další velkou výhodu patří bezchybnost testovacích nástrojů. Pokud jsou nástroje správně nakonfigurovány, provádí test vždy stejně a nevnáší do testu chybu, kterou by tam mohl zanést nepozorný tester.

Výhody podle R. Pattona [6] jsou:

- **Rychlost –** Automatické nástroje provádí testy několikanásobně rychleji, než je tomu u manuálního testování.
- **Efektivita –** Testovací nástroj může jeden scénář provést několikrát. Není tedy potřeba, aby tester musel opakovaně provádět jednotlivé kroky scénáře a může se věnovat jiné práci.
- **Správnost a přesnost –** Automatické nástroje provádí testy vždy stejně a bezchybně. Eliminují tedy riziko, že by se do testu dostala chyba z důvodu nepozornosti testera.
- **Neúnavnost –** Nástroje pracují bez přestávky tak dlouho jak je potřeba.

Mezi hlavní nevýhody patří neustálá údržba testů. Pokud je aktualizována jakákoliv část softwaru, nebo pokud je přidána nová funkcionalita musí být aktualizovány také testy. Další nevýhodou je nutnost vyčlenit testování speciální prostředí a připravit vhodná testovací data [16]. Psaní automatických testů vyžaduje zkušeného programátora nebo velmi zkušeného technického testera. Tvorba automatického testu trvá mnohem delší dobu než ruční testování. K jednotlivým testovacím případům se musí nejprve vytvořit scénář, který vzniká ručním testováním. Tyto scénáře jsou poté automatizovány.

S automatizací testů se též pojí termín paradox pesticidů. Tento termín popisuje, že pokud budeme software testovat pořád stejnými testy dokola, stane se testovaná část imunní vůči těmto testům. Je tedy potřeba vymýšlet nové testy, aby byla možnost nalézt nové chyby [17].

4.2 Proces automatizace testů

Pokud se u nějakého softwaru rozhodne o použití automatických testů, je vhodné postupovat podle jednotlivých kroků, které definují proces automatizace. To zajistí, že výsledný efekt nasazení automatických testů bude optimální.

4.2.1 Výběr nástroje

Prvním krokem v tomto procesu je správný výběr testovacího nástroje. Tento krok je nejdůležitějším a také nejtěžším krokem. V dnešní době se na trhu objevuje spousta nástrojů, kde každý se hodí na jiný typ softwaru. Záleží tedy hodně na technologii, na které je software vyvíjen [18]. Dále je nutné zhodnotit, zda využít nějaký placený nástroj, který bývá zpravidla robustnější nebo použít open source nástroj. To většinou závisí na velikosti produktu, který je vyvíjen, a pro který se plánuje použití automatických testů.

4.2.2 Definice rozsahu automatizace

Je vhodné určit si jakousi politiku, které testy se budou automatizovat, a které nikoliv. Vhodným kandidátem pro automatizaci je testovací případ neboli test case, který se ve značné míře opakuje nebo testuje důležitou funkcionalitu v rámci

výsledného produktu. Existuje několik kritérií, které pomáhají určit tento rozsah. Podle webové stránky Guru99 [18] jsou to tyto:

- Funkce, které jsou důležité pro výsledný software.
- Scénáře, které pracují s velkým množstvím dat.
- Funkce, které jsou společné pro více aplikací.
- Technická proveditelnost.
- Složitost testovacích případů.
- Možnost používat stejné testovací případy napříč prohlížeči.

4.2.3 Plánování, návrh a vývoj

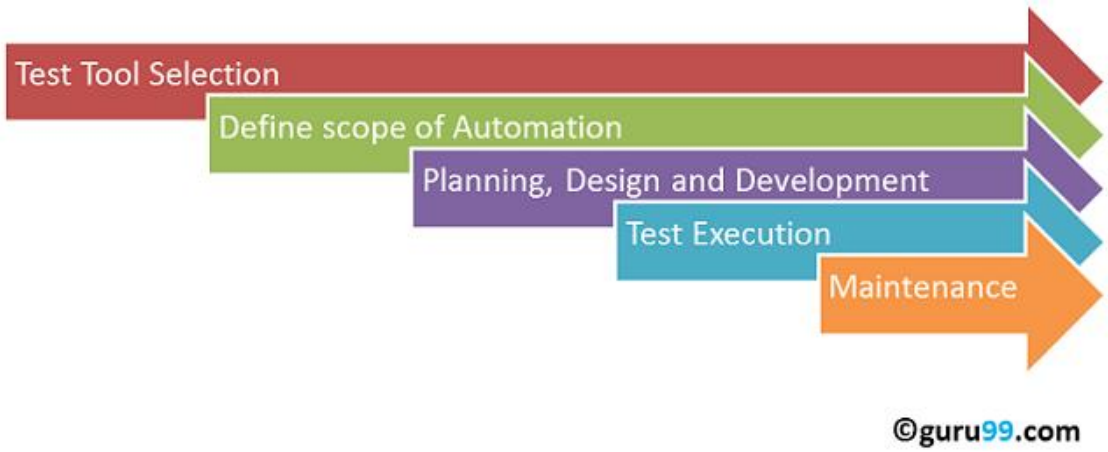
V tomto kroku přichází na řadu samotná tvorba automatizovaného testu. Před samotnou tvorbou skriptu je vhodné nejdříve provést manuální testování a jednotlivé kroky zaznamenat k danému testovacímu případu. V takovém záznamu je pak napsáno, jaké kroky musel tester učinit, aby se k testované funkcionalitě dostal, jakou funkcionalitu testoval a jaký tam očekával výsledek [18]. Takto popsaný testovací případ zajistí, že při jeho automatizaci nebude vynechána žádná důležitá část.

4.2.4 Vykonání testu

Jak už napovídá název, v této fázi probíhá samotné spouštění skriptů. Jakmile jsou provedeny, poskytují podrobné protokoly o průběhu testu a umožňují jejich následnou analýzu [18].

4.2.5 Údržba

Posledním krokem tohoto procesu je údržba existujících automatizovaných testů. V případě změny některé funkcionality je nutné opravit i testy [18].



Obrázek 4: Proces automatizace testů
Zdroj: [18]

4.3 Nástroje pro automatické testování

Bez automatického testování se dnes neobejde žádný projekt, u kterého je předpokládaná určitá kvalita. Nejenom že pomáhá vyvinout produkt o určité kvalitě, jelikož pomáhá odhalit chyby ve správný čas, ale také šetří čas a peníze. I když počáteční investice může být celkem vysoká u robustnějších produktů, určitě se to vyplatí.

Vytváření automatizovaných testů není jednoduché. Vyžadují zkušeného testera jak pro tvorbu testu, tak pro jeho následnou údržbu. Aby se tvorba testů co nejvíce zrychlila a zjednodušila, jsou nezbytné nástroje pro tvorbu testů. Při vývoji softwaru je celkem běžné, že se mění dříve implementované funkcionality. Je proto nutné na tyto změny reagovat i v jednotlivých testech. Tyto nástroje tedy umožnují nejenom samotnou tvorbu testů, ale také jednoduchou správu už existujících testů. Další nezanedbatelnou výhodou těchto nástrojů je analýza výsledků testů. Umožňují je zobrazit v několika přehledných formách jako jsou tabulky, grafy atd.

4.3.1 Kritéria pro výběr nástroje

Jak již bylo řečeno výběr nástroje je jedním z nejdůležitějších a nejtěžších kroků v procesu automatizace testů. Je nutné si stanovit kritéria, které od daného testovacího nástroje očekáváme. Těchto kritérii může být nepřeberné množství, ale v této práci je použito 6 nejběžnějších kritérii, které bývají zahrnuty při rozhodování. Podle průzkumu stránky DZone [19], kde bylo dotázáno více jak 2000 profesionálních testerů, to jsou tato kritéria:

- **Náklady na licence a podporu** – zahrnuje veškeré náklady, které jsou spojené se získáním a udržením podpory používaných nástrojů. Priorita tohoto kritéria hodně závisí na zkušenostech a schopnostech testovacího týmu a velikosti projektu, který testují. Open-source nástroje většinou vyžadují více zkušeností než komerční nástroje.
- **Dobré reporty z testů –** udává v jaké formě umí nástroj generovat výsledky testů. Nástroj, který neumí v rozumné míře zobrazovat výsledky, které je možné dále analyzovat, je víceméně nepoužitelný.
- **Školení, dokumentace –** udává v jaké míře existuje pro daný nástroj dokumentace nebo zda se pořádají různá školení. Priorita tohoto kritéria se u různých testovacích týmů může lišit. Některý tým upřednostní výbornou dokumentaci a školení před náklady
- **Podpora průběžné integrace a technologie DevOps –** průběžná integrace je proces, při kterém se slučuje nově napsaný kód s kódem v centrálním uložišti.
- **Požadovaná úroveň programovacích dovedností**
- **Požadovaná úroveň dovedností a zkušeností**

5 JMeter

JMeter od Apache Software Foundation je open-source desktopová aplikace napsaná v programovacím jazyce Java. Tudíž zde odpadá problém s kompatibilitou a JMeter je možné využívat na jakémkoliv operačním systému, který podporuje Javu.

JMeter byl primárně určený pro testování webových nebo FTP aplikací. V dnešní době umožňuje testovaní JDBC, webové služby, generické TCP spojení a nativní procesy operačního systému. Nabízí provádění různých druhů testů, jako jsou výkonové testy, zátěžové testy, stress testy, regresní testy nebo funkční testy [20].

5.1 Instalace

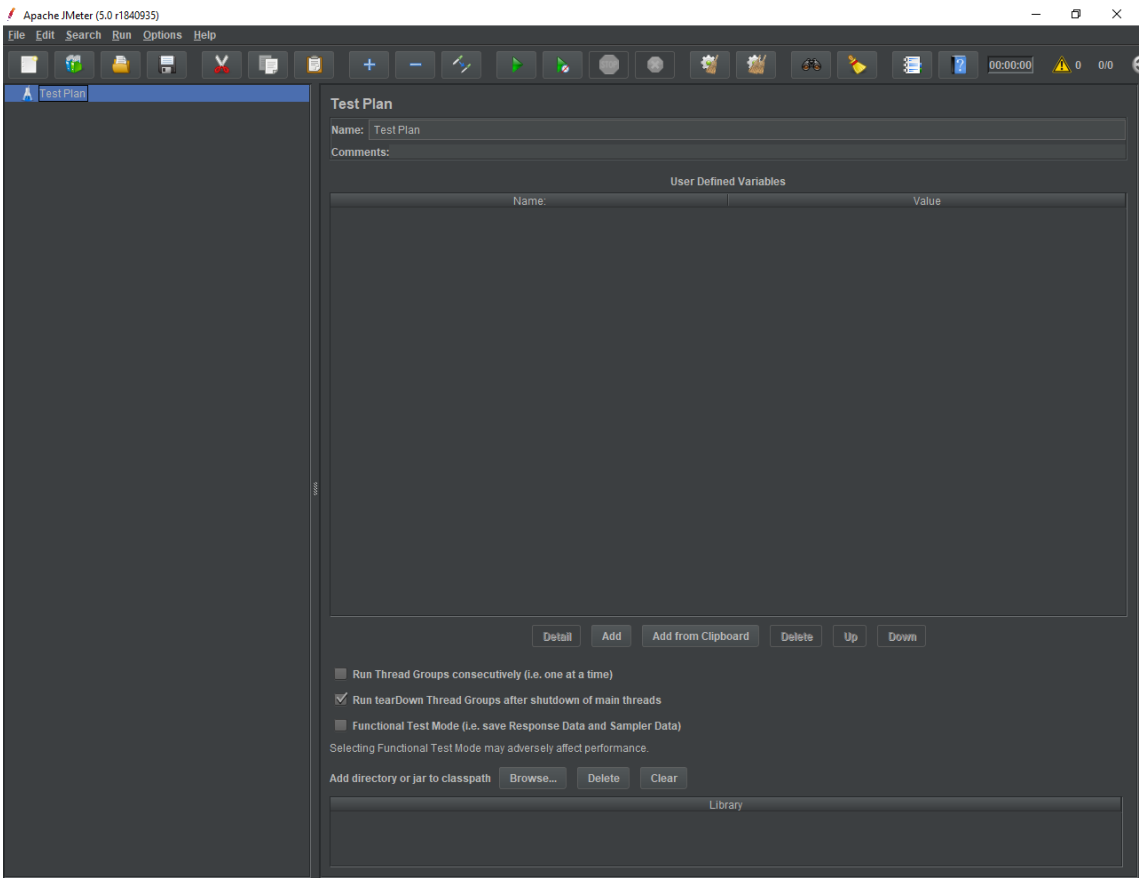
Jelikož JMeter je ryze Java aplikace, je nutné mít nainstalovaný Java Runtime Enviroment (JRE) nebo Java Developement Kit (JDK) v požadované verzi. Poté je možné spustit nástroj pomocí dávkových souborů, které spouští JMeter v různých režimech. Tyto soubory se nachází ve složce bin v balíčku, který je možné najít a stáhnout na oficiální stránkách JMeteru.

5.2 Uživatelské rozhraní

JMeter je možné spustit ve dvou hlavních režimech. Režim s grafickým uživatelským rozhraním nebo v režimu příkazové řádky. Existuje i několik způsobů, jak vytvořené testy spouštět, které jsou popsány dále. Používání grafického režimu se hodí spíše pro vytváření a debuggování testů než pro jejich spouštění.

Grafické prostředí je na pohled příjemné a přehledné. Samotné okno je rozdělené do třech hlavních částí. Na levé straně panel, který obsahuje test plan

konfigurační okno, které zabírá většinu prostoru, a nakonec lišta menu, která zastřešuje obě podokna.



Obrázek 5: JMeter uživatelské rozhraní

Zdroj: Vlastní zpracování

V levém panelu se ihned po spuštění nachází položka Test Plan. Test Plan může být jakýsi kontejner, který obsahuje jednotlivé testovací scénáře a testovací data. Nebo to může být kontejner, který obsahuje posloupnost kroků, které se provedou po spuštění tohoto Test Planu. Testovací plán může obsahovat jeden nebo více elementů, jako jsou [21]:

- **Thread Group** – definuje počet vláken, ve kterých test poběží.
- **Logic Controller** – umožňuje definovat logiku, se kterou bude JMeter rozhodovat o tom, kdy mají být požadavky odesílány.
- **Configuration Element** – slouží k definici různých konfigurací
- **Sampler** – definuje požadavky, které jsou odesílané na server. Zpracovávají se v pořadí, v kterém jsou definované ve stromu.

- **Timer** – udává, v jakém časovém sledu budou odesílány jednotlivé požadavky.
- **Listener** – zprostředkovávají přístup k informacím, které JMeter shromažďuje. Jednoduše řečeno poskytuje reporty.
- **Assertions** – slouží k ověření dat, které aplikace vrací.

V konfiguračním panelu se poté provádí konfigurace samotného testovacího plánu a jeho jednotlivých prvků. U testovacího plánu je možné změnit jméno, nadefinovat jednotlivé proměnné a nastavit vlastnosti, které určují chování test planu podle požadavků.

5.3 Práce s nástrojem

V této podkapitole jsou popsány jednotlivé možnosti vytváření testu, jeho možná úprava a analýza výsledků.

5.3.1 Implementace testu

Pro vytvoření testu existují dva způsoby. Buď je možné test vytvořit manuálně přidáváním jednotlivých elementů (např. HTTP request) nebo vytvořit test pomocí nahrávání.

Při vytváření testu manuálně je první krok vytvořit tzv. Thread Group, která vytváří skupinu pro ostatní elementy. Klikem pravého tlačítka myši na Test plan se vyvolá kontextové menu, kde se vybráním *ADD→Threads(Users)→ Thread Group* Thread Group vytvoří. Ostatní elementy se přidávají stejným způsobem. U takto vytvořené skupiny je nutné nastavit počet virtuálních uživatelů, kteří budou přistupovat k testované aplikaci. Dále je zde možnost určit dobu, jakou bude trvat, než všichni uživatelé odešlou svůj požadavek a počet opakování připojení uživatele. Pokud je Thred Group vytvořená a nakonfigurovaná, nezbývá nic jiného než začít přidávat jednotlivé HTTP požadavky, požadované listenery a ostatní potřebné elementy.

V případě vytváření testu nahráváním existuje několik možností. První možnost je použít šablonu a recorder vestavěný přímo v JMeteru. V horní liště pod záložkou File je položka Templates, kde se vybráním šablony Recording JMeter

předpřipraví základní struktura test planu. Testerovi poté stačí zapnout nahrávání, které nastartuje proxy server na daném portu a vytvoří certifikát. Tento certifikát je potřeba přidat mezi důvěryhodné certifikáty v prohlížeči. Mimo toho je nutné také nastavit adresu proxy serveru. Pokud jsou tyto kroky splněny, může tester na webové aplikaci začít provádět manuální test a všechny jeho kroky jsou zaznamenávány do test planu v JMeteru. Pokud tester požaduje zachytávání pouze určitých http požadavků, může si nastavit tzv. pattern, který mu tyto http požadavky vybere.

Další možností, jak test nahrát je využití aplikací třetích stran. Mezi oblíbené patří desktopová aplikace BadBoy nebo rozšíření do prohlížeče Google Chrome. Oba tyto nástroje umí nahrávat jednotlivé kroky a následně provést export do formátu jmx, což je formát testovacích skriptů pro JMeter.

5.3.2 Úprava testovacího skriptu

Jak již bylo řečeno v předešlých kapitolách, funkcionality aplikací se v průběhu vývoje i v průběhu údržby mohou měnit a je proto nutné udržovat i jednotlivé testy, aby byly schopné neustále testovat potřebné funkcionality.
Díky tomu, že JMeter disponuje grafickým rozhraním, a že testovací plány jsou složeny z jednotlivých elementů, není editace a s tím spjatá údržba žádný velký problém. Jednoduše se vyberou potřebné elementy a provedou se u nich potřebné změny.

Editace je také nutná v případě tvorby testu nahráváním, jelikož se tímto způsobem dostane do testu spoustu http požadavků, které není nutné zahrnout do testu.

5.3.3 Spuštění testu

Jakmile je test připraven, následuje samotné spuštění testu. Existují dva způsoby, jak vytvořený test spustit. Prvním z nich je spuštění testu přímo v grafickém uživatelském režimu. Tato možnost je vhodná v případě vytváření testu, jeho debuggování nebo spouštění testů pro jednotky vláken. V případě zátěžových testů je lepší způsob spouštět testy v příkazové řádce pomocí příkazu uvedeného v následující tabulce.

Tabulka 3: Spuštění JMeter testu v non-gui režimu

jmeter – n –t test.jmx -l testresults.jtl
-n spuštění v non-gui režimu
-t cesta k jmx souboru
-l cesta k logovacímu souboru

Zdroj: Vlastní zpracování

5.3.4 Analýza výsledků

Samotná analýza výsledků je jedna z nejdůležitějších fází testování. Umožnuje testerům vytvořit závěry z průběhu testu. JMeter umožňuje reprezentovat výsledky v několika formách. A to pomocí tzv. Listenerů, které poskytují přístup k informacím, které JMeter shromažďuje. Formy reprezentace těchto informací mohou být např. strom, tabulka, graf, nebo výstup do logovacího souboru.

5.4 Zhodnocení nástroje

JMeter je velmi silný nástroj, s příjemným uživatelským rozhraním, které umožňuje tvořit testy i bez znalosti programovacích jazyků. Hodí se především na zátěžové nebo stress testy. JMeter nemusí sloužit pouze na testování, ale může být použit i pro odesílání požadavků na server. Velkou výhodou nástroje je licence, protože je zdarma.

6 IBM Rational Functional Tester (RFT)

Rational Functional Tester od společnosti IBM je komerční, objektově orientovaný nástroj pro automatické testování. Je schopný provádět funkční, regresní nebo data driven testování. Data driven testování je metodika testování, která využívá tabulku testovacích vstupů a ověřitelných výstupů. RFT podporuje širokou škálu aplikací a protokolů jako HTML, Java, .Net, SAP, Dojo, Visual Basic a mnoho dalších [22].

Díky možnosti testování HTML 5 a různých UI frameworků, umí RFT od verze 9.2 také testovat uživatelské rozhraní na desktopu nebo mobilních zařízeních. RFT je multiplatformní nástroj a tudíž běží na operačních systémech Windows, Linux i Mac OS. Ovšem testování založené na HTML 5, a tudíž i testování uživatelského rozhraní, je dostupné pouze na Windows a Linux [22].

6.1 Instalace

Instalace probíhá klasickým způsobem. Po stažení balíčku nebo vložení instalačního media je nutné spustit průvodce instalací. Během několika kroků uživatel zvolí jednotlivé balíčky, místo instalace a jazyky, co mají být nainstalovány.

Pokud se v počítači vyskytuje IDE Eclipse, je možné přidat RFT jako rozšíření tohoto prostředí. Není to ovšem nutnost. Společně s instalací RFT se provede také instalace IBM Instalation Manager, pomocí kterého je možné provádět aktualizace jednotlivých IBM produktů.

6.2 Uživatelské rozhraní

RFT disponuje přehledným a jednoduchým uživatelským rozhraním. Hned na první pohled je zřejmé, že RFT je postavené na vývojovém prostředí Eclipse, jelikož uživatelské prostředí je téměř stejné jako u Eclips.

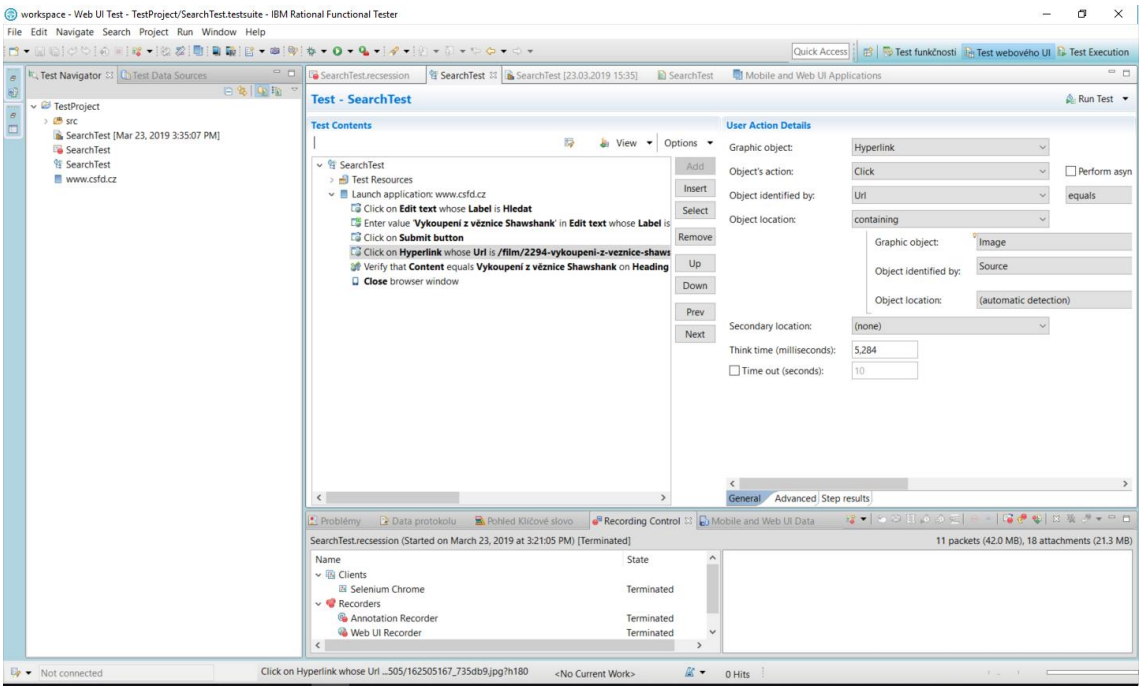
Pod horním panelem, kde jsou jednotlivé nástroje a funkce, se nachází samotná pracovní plocha, která je složená z panelů. Tyto panely je možné přidávat a přesouvat dle potřeby. Je tedy možné si pracovní plochu uspořádat podle osobních preferencí. Součástí horní, nástrojové lišty jsou přepínače mezi jednotlivými perspektivami. Perspektivy jsou předdefinovaná pracovní prostředí, která obsahují

potřebná okna a nástroje pro daný účel, například pro testy funkčnosti, testování webového UI nebo samotné spouštění testů.

Na levé straně se nejčastěji nachází navigační okno, které obsahuje stromovou strukturu projektu včetně jednotlivých testových elementů.

V prostřední části je hlavní okno, které slouží pro zápis testového skriptu v jazyce Java, nebo pro konfiguraci nahraného skriptu.

Pod hlavním oknem je pomocné okno, které zobrazuje doplňkové informace, jako jsou například chybové hlášky, výpisy do konzole a jiné.



Obrázek 6: RFT Uživatelské rozhraní

Zdroj: Vlastní zpracování

6.3 Práce s nástrojem

Před tvořením testovacích skriptů je potřeba provést nastavení testovacího prostředí. V první řadě je nutné zkontrolovat, zda je dostupná podporovaná verze JRE, a také zda je dostupný podporovaný webový prohlížeč. Ve verzi 9.2.0 má RFT automaticky povolené prostředí pro funkční testování. Tudíž je možné jít nahrávat testovací skripty bez manuálního povolování jednotlivých komponent.

6.3.1 Implementace testu

Prvním krokem při vytváření nového testu je vytvoření samotného testového projektu, kam se následně budou přidávat jednotlivé testové skripty. Vytvoření projektu se provede přes panel nástrojů *File→New→Project.* V přehledném průvodci se provede konfigurace nového projektu. Je nutné zvolit typ projektu, jako funkční testování, web UI test a mnoho dalších. Jedna z možností je připojit nový projekt k už existujícímu projektu.

Po vytvoření projektu je možné se přesunout na tvoření jednotlivých testovacích skriptů. Existuje více možností, jak testovací skript vytvořit. Pro testery, kteří neumí pracovat s programovacím jazykem Java, je jistě nejjednodušším způsobem vytvořit test nahráním uživatelských akcí v testované aplikaci. Tyto akce jsou dále převedené do zdrojového kódu a je možné je dále editovat.

Druhou možností je použití objektové mapy, kam se vytváří jednotlivé objekty, s kterými uživatel pracuje. Tyto objekty tedy reprezentují prvky v GUI, jako jsou tlačítka, textové pole a jiné. Výhodou objektové mapy je, že vzniká bohatá sada API rozhraní. Sada těchto rozhraní poskytuje možnost vytvořit testovací Framework, který zamezí výskytu duplicit a značně zjednoduší práci testerům na složitých projektech.

6.3.2 Verifikační body

Při vykonávání testu je potřeba kontrolovat, jestli došlo k určité akci nebo jestli stav ovládacího prvku nebo nějakého objektu má požadovaný stav. K tomu slouží verifikační body. Vytvoření verifikačního bodu je možné více způsoby.

V případě vytvoření testového skriptu nahrávání stačí otevřít samotný test, kde jsou vyobrazené jednotlivé kroky scénáře. Za požadovaný krok se vloží verifikační bod, u kterého je potřeba vybrat typ objektu, identifikaci tohoto objektu, popřípadě umístění a vlastnost společně s požadovanou hodnotou.

Verifikační bod je také možné přidat přímo k objektu v testovém skriptu. Stačí nalézt požadovaný objekt v hierarchii objektů a nastavit mu kontrolovanou vlastnost a požadovanou hodnotu této vlastnosti.

6.3.3 Parametry v testovém skriptu

Každý testový skript potřebuje pro svůj správný průběh hodnoty, které v normálním případě uživatel vyplňuje do textových polí nebo hodnoty pro ověření určité vlastnosti. V RFT se k práci s těmito daty používá tzv. datapool. Každý vytvořený skript má svůj vlastní privátní datapool. Navíc je možné vytvořit sdílený datapool, který může používat několik skriptů najednou.

Pro vytvoření sady dat je vhodné použít nějaký textový nebo tabulkový editor. V tomto editoru se vytvoří CSV soubor, který obsahuje jednotlivé hodnoty společně s názvy proměnných. Nakonec je potřeba vytvořený CSV soubor importovat do RFT.

6.3.4 Úprava testovacího skriptu

V případě úpravy existujícího skriptu se naskytují dvě možnosti, jak takový skript upravit. Možnost úpravy se odvíjí od způsobu vytvoření testového skriptu. V případě, že je test vytvořený nahráváním je možné požadovanou část nahrát znovu se změněným chováním.

V druhém případě, kdy je skript napsán manuálně v programovacím jazyce Java, se editace provádí úpravou jednotlivých objektů a samotného skriptu. V tomto případě může být úprava velmi rychlá, což je velkou výhodou při vytváření takovéhoto API a testového frameworku. Při změně funkcionality není nutné nahrávat změněné části testů znovu, ale stačí upravit požadované objekty. Což pro zkušenějšího testera, který disponuje znalostí programovacího jazyka Java, není žádný velký problém.

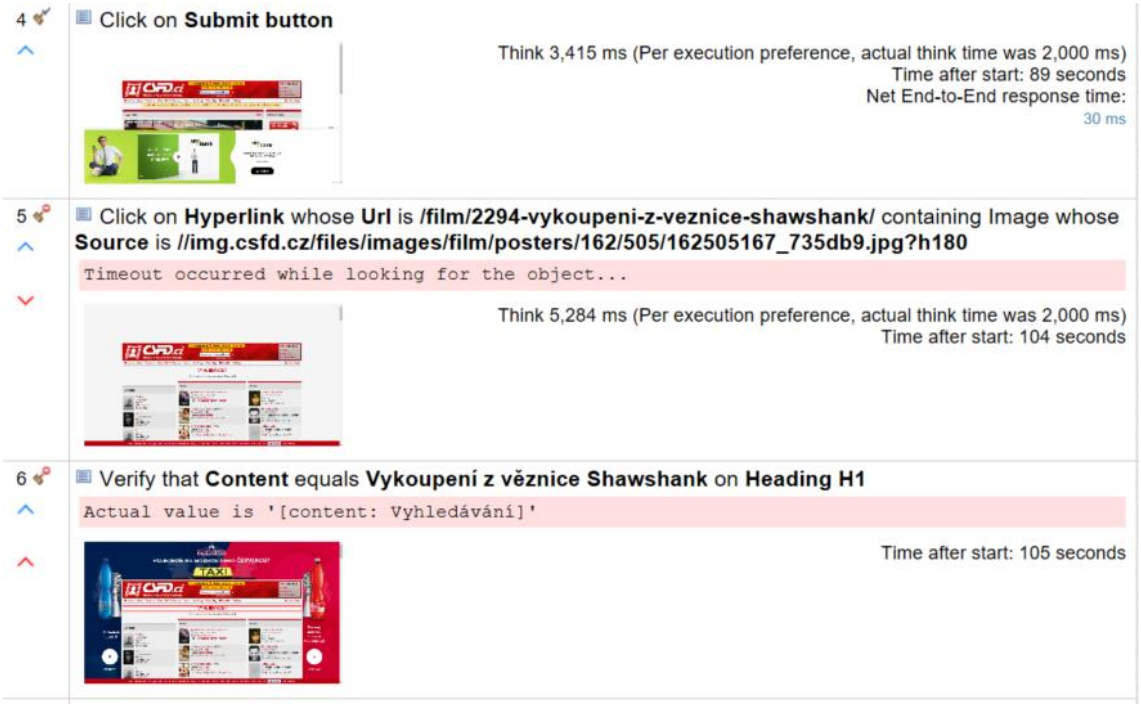
6.3.5 Spuštění testu

Při spuštění testu se otevře konfigurace běhu testu. V případě testování webové aplikace je zde možné zvolit například webový prohlížeč, ve kterém se test bude provádět. Po provedení potřebné konfigurace se otevře webový prohlížeč s testovanou webovou aplikací a začnou se provádět jednotlivé kroky testového scénáře. Po skončení testu se zobrazí logy a výsledky provedeného testu.

6.3.6 Analýza výsledků

Po dokončení testů se otevřou jednotlivá okna s údaji a statistikami právě provedeného testu. Jsou zde například vyobrazená data o jednotlivých krocích v testovém scénáři společně s časovými údaji nebo doba trvání celého testu. RFT také shromažďuje údaje o době spouštění nebo vypínání aplikace a vytváří z toho maximální, minimální a průměrnou dobu, které zobrazuje v přehledných sloupcových grafech.

V dalším okně je pomocí koláčového grafu reprezentován poměr úspěšných a neúspěšných kroků. Z toho grafu je možné přejít k výpisu jednotlivých kroků. U každého kroku je uveden čas v sekundách, kdy byl krok proveden. Tento čas se měří od startu aplikace. Pokud krok selhal, je zde zaznamenáno, z jakého důvodu došlo k selhání.



Obrázek 7: Výsledek testu v Rational Functional Tester

Zdroj: Vlastní zpracování

6.4 Zhodnocení nástroje

Rational Functional Tester je robustní nástroj, který se hodí primárně pro velké projekty. Výhodou je vytváření testových skriptů pomocí nahrávání uživatelských akcí. V případě složitějších projektů je však nutné začít používat objektovou mapu a vytvářet si jednotlivé objekty a popřípadě vytvořit testovací framework. Výhody a schopnosti tohoto nástroje jsou ovšem vykoupeny vysokou cenou.

7 Sahi

Sahi je poměrně neznámý nástroj napsaný v jazycích Java a Javascript určený pro automatizaci testů. I přesto je to silný nástroj primárně určený pro testování webových aplikací. Díky doplňkům je schopný vytvářet testy i pro desktopové aplikace nebo pro aplikace postavené na mobilní platformě Android.

Sahi je dostupný v bezplatné open source verzi nebo v komerční proprietární verzi. Bezplatná verze obsahuje základní nástroje, které jsou dostačující pro malé a jednoduché aplikace. Umožňuje testovat pouze webové aplikace ve dvou prohlížečích, nepodporuje pokročilé logy a disponuje pouze elementárním nástrojem pro editaci skriptů. Komerční verze je na tom o několik tříd lépe a je vhodná i pro komplexnější aplikace.

V následujících podkapitolách je popisován pouze Sahi Pro což je komerční proprietární verze.

7.1 Instalace

Jelikož je Sahi napsaný v jazyce Java je, nutné mít nainstalovaný JRE ve verzi alespoň 1.6. V případě, že je JRE připravený, je možné přejít k instalaci samotného nástroje. Po stažení Sahi knihovny a knihoven doplňků pro desktopové a mobilní aplikace probíhá instalace pomocí jednoduchého instalačního průvodce. Během instalace se volí balíčky, které mají být nainstalovány.

Po dokončení instalace je možné vytvořit skript, který obsahuje veškeré informace o instalaci. Tento skript se používá například na ostatních zařízeních, aby se nemusel procházet instalační průvodce na každém zařízení zvlášť. Instalace s použitím tohoto skriptu se poté provádí pomocí následujícího příkazu.

Tabulka 4: Instalace Sahi pomocí instalačního skriptu

java – jar install_sahi_pro_xxx.jar install_script.xml

Zdroj: Vlastní zpracování

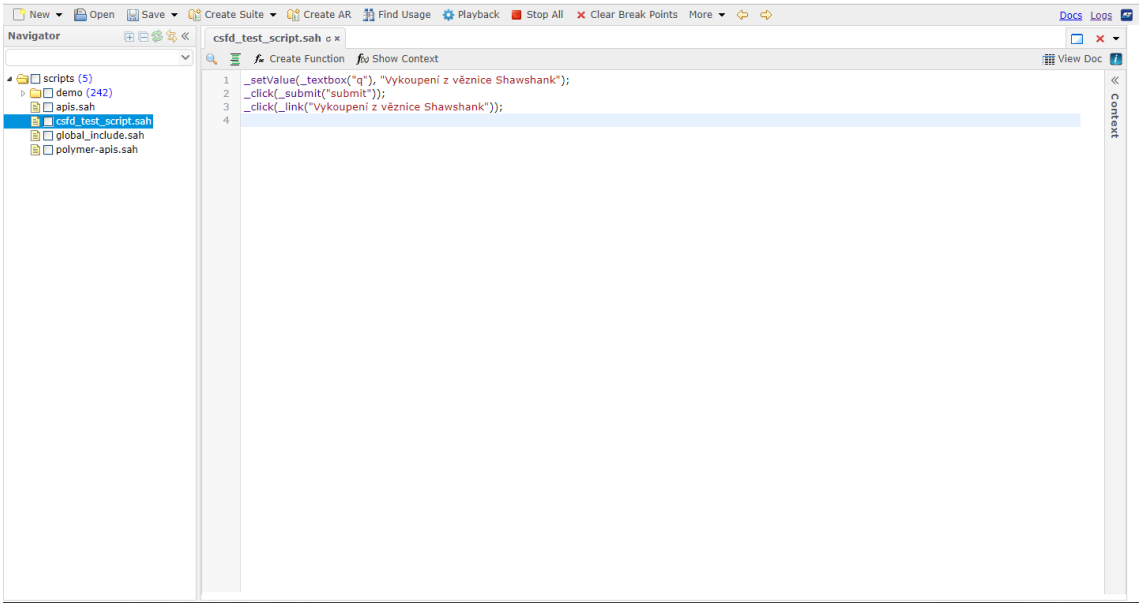
7.2 Uživatelské rozhraní

Ke všem nástrojům se přistupuje pomocí webového prohlížeče. Sahi tedy může běžet na lokálním zařízení testera nebo na serveru.

Po spuštění Sahi se objeví panel Sahi Dashboard. Odtud se přistupuje ke konfiguraci, skriptům, editoru, logům a také k dokumentaci. Dále jsou zde jednotlivé pohledy, ve kterých je možné vytvářet a nahrávat testy. Jsou zde tedy veškeré dostupné prohlížeče. Pokud jsou v Sahi nainstalované doplňky pro testování desktopových a mobilních aplikací, jsou dostupné i tyto pohledy.

Ve zvoleném prohlížeči se pomocí kombinace klávesy alt a dvojitým klikem levého tlačítka myši stránku zobrazí kontrolér. Ten poskytuje prostředí pro vytváření, nahrávání a jednoduché přehrávání skriptů.

Pro editaci a přehrávání vytvořeného skriptu je vhodné využít editor, který má velmi jednoduché a přehledné prostředí s lištou nástrojů, navigačním oknem a pracovní plochou pro editaci skriptů.



Obrázek 8: Uživatelské prostředí Sahi editoru
Zdroj: Vlastní zpracování

7.3 Práce s nástrojem

Práce s nástrojem Sahi spočívá v neustálém přepínání mezi jednotlivými nástroji pro nahrávání, editaci a analýzu výsledků. Což se může zdát na první pohled zmatečné, ale na druhou stranu má každá oblast svůj jasně definovaný, přehledný prostor.

7.3.1 Implementace testu

Implementace testu začíná vytvořením nového testového skriptu. Jakmile je skript připraven, je možné přejít k nahrávání samotného testu. To se provede jednoduchým stiskem tlačítka record v kontroléru. Od tohoto okamžiku jsou zaznamenávány a ukládány veškeré kroky uživatele, které provede ve webové aplikaci.

Sahi kontrolér také umožnuje identifikaci jednotlivých elementů. Pomocí podržení klávesy Ctrl a najetím ukazatele na požadovaný element se zobrazí daný identifikátor. Sahi k identifikaci nepoužívá klasický jazyk XPath, ale identifikuje elementy pomocí jejich atributů (viditelný text, název, id, třída). Identifikátory jsou pak srozumitelnější. Jistou výhodou této identifikace je, že identifikátory jsou schopni přečíst i uživatelé, kteří neznají jazyk XPath. Nevýhodou je, že ne všechny identifikace lze udělat pomocí atributů, jelikož existují elementy, které žádný atribut nemají anebo mají tento atribut dynamicky generovaný.

7.3.2 Verifikační body

Verifikační body je možné vkládat už během nahrávání samotného testového skriptu. Vložení je poměrně jednoduché. Stačí identifikovat element, který je potřeba kontrolovat. Identifikace se provede podržením klávesy Ctrl a najetím ukazatelem myši na požadovaný element. Po identifikaci stačí pouze nastavit požadovanou hodnotu, kterou by element měl mít a pomocí tlačítka Assert se vloží verifikační bod do skriptu. Další možností je verifikační bod dopsat do už nahraného skriptu. To se provede pomocí jednotlivých ověřovacích funkcí.

7.3.3 Úprava testovacího skriptu

Po nahrání testu je velmi žádoucí otevřít skript v editoru a provést na něm alespoň základní úpravy. Nahrané kroky je vhodné obalit do jednotlivých funkcí, které mohou být znovu použitelné. Díky použití funkcí se kroky rozdělí do jednotlivých skupin, kde každá skupina může být ověřována pomocí verifikačních bodů. Samotné verifikační body mohou být také obaleny funkcí a tím použité na více místech. Testový skript se těmito kroky stane více přehledný.

Jednotlivé skripty mohou být spouštěné samostatně nebo se mohou přidat do Test Suit, což není nic jiného než skupina umožňující spouštění více skriptů najednou.

7.3.4 Spuštění testu

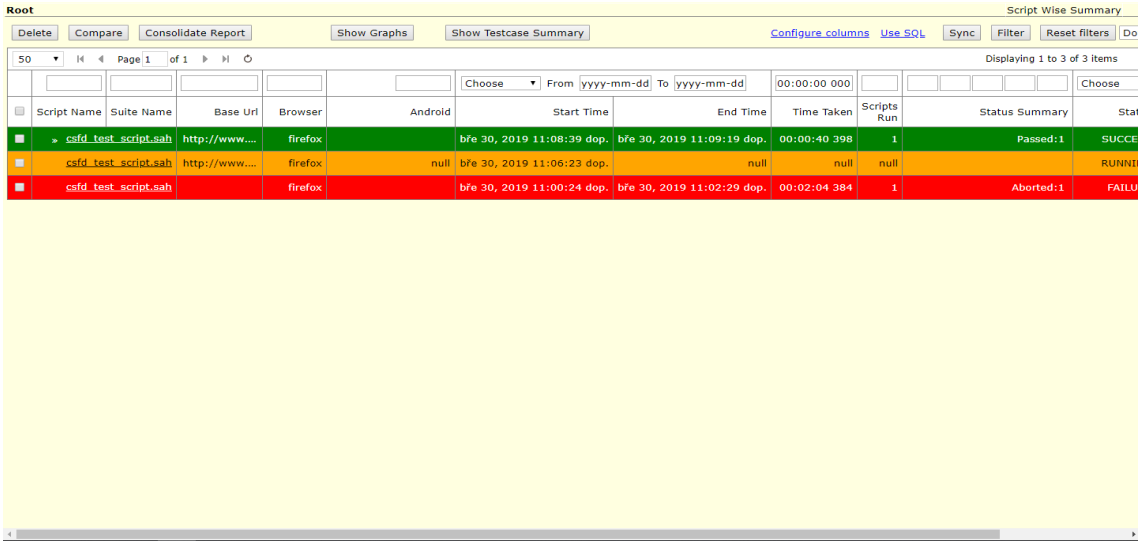
Když je test nahraný a upravený, je na řadě spuštění testu. Test se spouští buď přímo z kontroléru nebo z editoru. Lepší volbou je rozhodně použití editoru, jelikož umožňuje podrobné nastavení spuštění.

Před spuštěním je potřeba vyplnit startovací URL adresu, což je adresa webové aplikace. Dále se zde volí prohlížeč, kde se test provede, počet vláken, které poběží paralelně nebo e-mail kam bude zaslaný výsledek testu.

Test lze spustit v klasickém režimu nebo v režimu debuggování, což se hodí hlavně při prvotním vývoji testu.

7.3.5 Analýza výsledků

Po dokončení testu se může uživatel přesunout do nástroje Logs. Zde jsou zobrazeny všechny vykonané testy a jejich výsledky v tabulce nebo grafech. Výsledky jsou zobrazené v tabulce nebo v grafech. Jednotlivé testy je možno filtrovat nebo porovnávat.



Obrázek 9: Výsledky testů v Sahi
Zdroj: Vlastní zpracování

7.4 Zhodnocení nástroje

Sahi se na první pohled může jevit jako nějaký druhořadý nástroj vhodný maximálně pro jednoduché aplikace menších rozměrů. Rozhodně tomu tak není, jedná se o komplexní nástroj. Navíc je Sahi vhodný i pro testery, kteří nemají velké zkušenosti s programováním.

8 Porovnání nástrojů

Každý z uvedených nástrojů byl otestován implementací jednoduchého testu uživatelského rozhraní. Na základě toho je provedeno následující porovnání.

8.1 Použitelnost nástroje

Každý z nástrojů uvedených v této práci se sice používá pro vytváření automatizovaných testů a pro práci s nimi, nicméně každý z nástrojů se může hodit pro jiný druh testů.

JMeter je vhodný především pro provádění zátěžových nebo stress testů. Pracuje na protokolové vrstvě, a tudíž se příliš nehodí na funkční testy nebo testy UI. Nicméně i tyto testy umožňuje, ale výsledné skripty mohou být příliš složité a je vhodnější použití jiného nástroje. Mimo testování lze použít JMeter i pro odesílání požadavků na server, tudíž na debuggování. Nespornou výhodou tohoto nástroje a jistým důvodem proč je tento nástroj tak oblíbený, je bezplatná licence. Hodí se tedy i na projekty jejichž rozpočet je omezený a není možné vložit velké finanční prostředky do testování.

RFT neumí provádět zátěžové nebo stress testy. Pro provádění těchto testů je potřeba zakoupit Rational Performance Tester, který je na to určený. RFT Je určený primárně na realizaci funkčních testů a testů uživatelského rozhraní jak na desktopových počítačích, tak mobilních zařízeních. Jednou z velkých slabostí a nevýhod je, že RFT využívá poměrně hodně paměti a mohou vznikat problémy v případě provádění velkých testů. Nástroj je dostupný pouze v komerční verzi, kde cena začíná na 6 800 dolarů za uživatele na rok.

Nástroj Sahi umožňuje provádět funkční testy, testy uživatelského rozhraní, ale i zátěžové testy. Jeví se tedy jako nejkomplexnější nástroj z nástrojů uvedených v této práci. Výhodou je provádění testů pro jednotlivé prohlížeče. Jednotlivé testy je možné seskupovat do Test suitu a spouštět je tak najednou. Sahi je dostupný v bezplatné i komerční verzi. Bezplatná verze nabízí nahrávání a spouštění testů ve všech dostupných prohlížečích, pokročilá editace nebo logy jsou však dostupné až v komerční verzi. Včetně doplňků pro desktopové a mobilní zařízení se cena pohybuje okolo 1795 dolarů za uživatele na rok.

Tabulka 5: Porovnání použitelnosti nástrojů

JMeter	Rational Functional Tester	Sahi
+ zátěžové a stress testy + bezplatná licence - příliš složité testy uživatelského rozhraní	+ funkční a UI testování - velká náročnost na paměť - vysoká cena	+ zátěžové i funkční testy + bezplatná i komerční verze

Zdroj: Vlastní zpracování

8.2 Křivka učení

Křivka učení slouží k vizualizaci pokroků učení v průběhu času [23]. V případě, že se jedná o vysokou učící křivku, porozumění dané problematice vyžaduje nějaký čas. U nízké učící křivky je tomu naopak.

JMeter disponuje poměrně jednoduchým uživatelským rozhraním. Testeři tedy nemusí mít znalosti programování. Přesto, že uživatelské rozhraní je přehledné, obsahuje spoustu možností, které vyžadují nějakou dobu na porozumění. Implementace jednoduchého testu zabrala z uvedených nástrojů nejdelší dobu. Nástroj je tedy vhodnější spíše pro zkušenější testery než pro začátečníky, jelikož učící křivka je vysoká.

RFT využívá pro zápis běžné jazyky jako je Java nebo .NET. Vyžadují tedy alespoň základní znalost těchto jazyků. Test lze sice vytvořit pouze pomocí nahrávání, nicméně následná editace znovu nahráváním požadované části skriptu může být krkolomná. V případě, že uživatel zná programovací jazyk Java nebo .Net je učící křivka poměrně nízká. Na druhou stranu v případě neznalosti programovacích jazyků je velmi vysoká. Výhodou může být vytvoření vlastního frameworku pro daný projekt. To umožní rychlejší integraci nově příchozích a nezkušených testerů do projektu.

Sahi je vhodný i pro testery bez velkých zkušeností. Díky propracovanému nahrávaní a identifikaci jednotlivých elementů jsou skripty dobře čitelné. Sahi využívá vlastní jazyk, který je velmi podobný Javascriptu. Je tedy vhodná alespoň minimální znalost programování. Učící křivka u toho nástroje je ze všech zde

uvedených určitě nejnižší, a proto vytvoření testu včetně úpravy a analýzy výsledků zabralo několik minut.

Tabulka 6: Učící křivky nástrojů

JMeter	Rational Functional Tester	Sahi
Vysoká učící křivka	**Znalost Java, .NET** – nízká učící křivka **Neznalost Java, .NET** – vysoká učící křivka	Nízká učící křivka

Zdroj: Vlastní zpracování

8.3 Správa a znovupoužitelnost testových skriptů

Při vytváření testů je vhodné kroky, které se opakují ve více testech, seskupit a zpřístupnit i pro ostatní testy. Zpravidla to bývají kroky při přihlašování nebo odhlašovaní z aplikace.

V JMeteru lze znovupoužitelnost zajistit vytvořením Test Fragmentu. Do toho fragmentu se poté vkládají jednotlivé elementy. V případě, že je potřeba analyzovat celý fragment, provede se to napojením Listeneru přímo na požadovaný fragment.

V RFT je znovupoužitelnost zajištěná díky jednotlivým objektům, které mohou být používány ve více skriptech. Navíc umožňuje znovupoužití i samotných skriptů včetně předávání parametru.

V Sahi se lze jednotlivé kroky obalovat do funkcí, což znamená, že mohou být použity i v jiných testech. Do jednoho skriptu lze vložit i jiný skript, a to pomocí include funkce.

8.4 Integrace na jiné nástroje

Všechny zde uvedené nástroje umožňují integraci s nástrojem Jenkins. Jenkins je nejznámější nástroj pro průběžnou integraci. Provádí automatické vytváření a testování projektu, což vývojářům značně usnadňuje integraci změn [24].

RFT dále umožňuje integraci s nástrojem Rational Quality Manager, který slouží pro plánování, spouštění a správu testů.

Sahi může být integrovaný do externích nástrojů pomocí předdefinovaných URL volání.

8.5 Analýza výsledků

Analýza výsledků se jeví jako jedno z nejdůležitějších kritérií. Zde uvedené nástroje mají zhruba stejné možnosti. RFT a Sahi navíc umí pořizovat snímky obrazovky u jednotlivých kroků testu. Co se týče přehlednosti pro nezkušeného uživatele, tak nejlépe z toho vychází Sahi. Na druhé straně je JMeter, kde jednotlivé přehledy bývají trochu komplikované.

8.6 Konečné hodnocení

Je velmi složité určit, který nástroj je nejlepší. Vždy je potřeba vzít v potaz, k čemu je nástroj potřeba, jaké testy je potřeba provádět, s čím vším je potřeba nástroj integrovat a v neposlední řadě také velikosti projektu a finančních možnostech. Dalším velmi důležitým kritériem jsou lidé, kteří mají s nástrojem pracovat. Záleží na jejich zkušenostech, a také zda umí programovat nebo zda mají nějaké zkušenosti s daným nástrojem.

Ovšem podle výše uvedených kritérií vychází nejlépe Sahi, což může být překvapení, jelikož není tolik známý. Nicméně je možné ho zařadit mezi jeden z nejlepších. Ovládání je intuitivní a vhodné i pro nezkušeného testera bez znalosti programovacích jazyků. Poskytuje také velice přehledné reprezentování výsledků, možnost znovupoužít jednotlivé skripty a poměrně jednoduchou editaci.

9 Závěr

Cílem bakalářské práce bylo seznámit čtenáře s pojmy a postupy v oblasti testování softwaru, představit automatizované testování včetně jeho výhod a nevýhod a v poslední řadě nalézt, představit a porovnat nástroje pro automatizaci testů na základě určitých kritérií.

V první části bakalářské práce byl představen samotný cyklus testování, kde byly popsány jednotlivé kroky od analýzy požadavků až po shrnutí výsledků testování. Následovalo popsání druhů chyb, které by měl každý tester umět rozeznat, jelikož na základě toho je možné provést rychlou a správnou opravu. Dále byly představeny metody testování, včetně druhů testů, které je možné provádět. Následující kapitola pojednávala o samotné automatizaci testů. Na základě představených důvodů, proč je vhodné testy automatizovat, lze říci, že automatizaci by se měla klást dostatečná pozornost na každém větším projektu. Automatizace má však také své nevýhody, které je nutné vzít v potaz při rozhodování, zda začít automatizovat.

Druhá část byla věnována nástrojům pro automatizaci. Byl proveden průzkum trhu. Na základě průzkumu byly vybrány tři nástroje. U každého nástroje byla popsána samotná práce s nástrojem včetně implementace jednoduchého testu uživatelského rozhraní webové aplikace. Poslední část popisu nástroje se věnovala možnosti ověřování a zobrazovaní výsledků testů.

Na závěr bylo provedeno porovnání těchto nástrojů na základě kritérií jako je použitelnost nástroje, snadnost naučení, možnost znovupoužitelnosti testů, ale také integrace na jiné nástroje a možnosti analýzy výsledků.

Na základě získaných informací a provedeného porovnání nástrojů lze říci, že není možné jednoznačně určit, který nástroj je nejlepší. Každý se hodí na něco jiného a při výběru je nutné vzít v potaz spoustu kritérií, které se s každým projektem mění. Co však lze s jistotou říci, že testování se stalo nedílnou součástí životního cyklu vývoje softwaru. Tomuto odvětví se dnes věnuje nemalá pozornost, jelikož umožňuje udržet vysokou kvalitu u komplexních produktů.

10 Seznam obrázků

Obrázek 1: Životní cyklus testování 8
Obrázek 2: Nejčastější příčiny chyb 10
Obrázek 3: V model testovacího procesu 16
Obrázek 4: Proces automatizace testů 21
Obrázek 5: JMeter uživatelské rozhraní 24
Obrázek 6: RFT Uživatelské rozhraní 29
Obrázek 7: Výsledek testu v Rational Functional Tester 32
Obrázek 8: Uživatelské prostředí Sahi editoru 35
Obrázek 9: Výsledky testů v Sahi 37

11 Seznam tabulek

Tabulka 1: Výhody a nevýhody testování černé skřínky....13
Tabulka 2: Výhody a nevýhody testování bílé skřínky....14
Tabulka 3: Spuštění JMeter testu v non-gui režimu....27
Tabulka 4: Instalace Sahi pomocí instalačního skriptu....34
Tabulka 5: Porovnání použitelnosti nástrojů....40
Tabulka 6: Učící křivky nástrojů....41

Seznam použité literatury

[1] SHARMA, Lakshay. Software testing. *TOOLSQA* [online]. 2016 [vid. 2018-08-13]. Dostupné z: http://toolsqa.com/software-testing/software-testing/

[2] Software testing. *Wikipedia* [online]. 2018 [vid. 2018-07-31]. Dostupné z: https://en.wikipedia.org/wiki/Software_testing.

[3] Software Testing Life Cycle STLC. *Software Testing Class* [online]. 2018 [vid. 2018-07-31]. Dostupné z: https://www.softwaretestingclass.com/software-testing-life-cycle-stlc/.

[4] STLC - Software Testing Life Cycle. *Guru99* [online]. 2018 [vid. 2018-07-31]. Dostupné z: https://www.guru99.com/software-testing-life-cycle.html

[5] Software bug. *Wikipedia* [online]. 2001 [vid. 2018-08-02]. Dostupné z: https://en.wikipedia.org/wiki/Software_bug

[6] PATTON, Ron. *Testování*. Praha: Computer Press, 2002. ISBN 80-7226-636-5.

[7] 7 Types of Software Errors That Every Tester Should Know. *Software Testing Help* [online]. 2018 [vid. 2018-08-02]. Dostupné z: https://www.softwaretestinghelp.com/types-of-software-errors/

[8] KITNER, Radek. Přehled testovacích technik. *Kitner* [online]. 2015 [vid. 2018-08-13]. Dostupné z: https://kitner.cz/testovani_softwaru/prehled-testovacich-technik/

[9] STOCKDALE, Dayana. The Top Black Box Testing Techniques. *Testlio* [online]. 2017 [vid. 2018-08-07]. Dostupné z: https://testlio.com/blog/top-black-box-testing-techniques/

[10] HLAVA, Tomáš. Testování bílé a černé skříňky (white box, black box, grey box). *Testování softwaru* [online]. 2011 [vid. 2018-08-07]. Dostupné z: http://testovanisoftwaru.cz/tag/seda-skrinkaik/

[11] HLAVA, Tomáš. Fáze a úrovně provádění testů. *Testování softwaru* [online]. 2011 [vid. 2018-08-13]. Dostupné z: http://testovanisoftwaru.cz/tag/integracni-testovani/#integration

[12] ZEMEK, Petr. Proč rozlišovat jednotkové a integrační testy. *O věcech, které mě baví* [online]. 2015 [vid. 2018-08-13]. Dostupné z: https://cs-blog.petrzemek.net/2015-04-18-proc-rozlisovat-jednotkove-a-integracni-

testy

[13] What is V Model in software testing and what are advantages and disadvantages of V Model. *Testingfreak* [online]. 2016 [vid. 2018-09-10]. Dostupné z: http://testingfreak.com/v-model-software-testing-advantages-disadvantages-v-model/

[14] KITNER, Radek. Regresní testy. *Kitner* [online]. 2015 [vid. 2018-08-18]. Dostupné z: https://kitner.cz/slovnik/regresni-testy/

[15] Types of Software Testing. *Geeks for geeks: A computer science portal for geeks* [online]. 2018 [vid. 2018-08-18]. Dostupné z: https://www.geeksforgeeks.org/types-software-testing/

[16] HLAVA, Tomáš. Automatizované testování. *Testování softwaru* [online]. 2016 [vid. 2018-09-10]. Dostupné z: http://testovanisoftwaru.cz/automatizovane-testovani/

[17] GROVER GARG, NISHI. Pesticide Paradox in Software Testing. *Testwithnishi* [online]. 2015 [vid. 2019-03-17]. Dostupné z: https://testwithnishi.com/2015/01/03/pesticide-paradox-in-software-testing/

[18] AUTOMATION TESTING Tutorial: What is, Process, Benefits & Tools. *Guru99* [online]. 2013 [vid. 2019-03-03]. Dostupné z: https://www.guru99.com/automation-testing.html

[19] NGUYEN, Vu. Top Priorities in Selecting Automated Testing Tools. *DZone: DevOps Zone* [online]. 2018 [vid. 2019-03-03]. Dostupné z: https://dzone.com/articles/top-priorities-in-selecting-automated-testing-tool

[20] STEVENS, Sander. Apache JMeter - Open Source Functional and Load Testing Tool. *Methods & Tools Software Development Magazine* [online]. 2010 [vid. 2019-03-05]. Dostupné z: http://www.methodsandtools.com/tools/tools.php?jmeter

[21] Elements of a Test Plan. *Apache JMeter* [online]. [vid. 2019-03-17]. Dostupné z: https://jmeter.apache.org/usermanual/test_plan.html

[22] Rational Functional Tester overview. *IBM Knowledge Center* [online]. 2018 [vid. 2019-03-23]. Dostupné

z: https://www.ibm.com/support/knowledgecenter/SSJMXE_9.2.1/com.ibm.rational.test.ft.doc/topics/IntrotoRobotJ.html

[23] ROUSE, Margaret. Learning curve. *WhatIs.com* [online]. 2016 [vid. 2019-04-02]. Dostupné z: https://whatis.techtarget.com/definition/learning-curve

[24] What is Jenkins? | Jenkins For Continuous Integration | Edureka. *edureka!* [online]. 2018 [vid. 2019-04-02]. Dostupné z: https://www.edureka.co/blog/what-is-jenkins/

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2018/2019

Studijní program: Aplikovaná informatika
Forma: Prezenční
Obor/komb.: Aplikovaná informatika (ai3-p)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Preisler Jan	Klácelova 3, Česká Třebová	I1600595

TÉMA ČESKY:

Porovnání nástrojů pro automatizaci testů v rámci vývoje SW

TÉMA ANGLICKY:

Comparison of tools for automation of tests in SW development

VEDOUCÍ PRÁCE:

doc. Ing. Vladimír Bureš, Ph.D., MBA - KIT

ZÁSADY PRO VYPRACOVÁNÍ:

Seznámení čtenáře s problematikou testování softwaru. Porovnání vybraných nástrojů v oblastech poměr cena/použitelnost nástroje, snadnost naučení, spravovatelnost, schopnost integrace na další nástroje. Zavedení testovacích scénářů do vybraných nástrojů.

SEZNAM DOPORUČENÉ LITERATURY:

Testování softwaru - Ron Patton
Just Enough Software Test Automation - Daniel J.Mosley, Bruce A. Posey
https://www.istqb.org

Podpis studenta: ..... **Datum:** .....

Podpis vedoucího práce: ..... **Datum:** .....