UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Testování webových služeb

Zdeňka Repáňová

Bakalářská práce
2014

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2013/2014**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Zdeňka Repáňová**
Osobní číslo:	**I10189**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Testování webových služeb**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Cílem práce je návrh a optimalizace procesu testování webových služeb, porovnání funkcionality a použitelnosti testovacích nástrojů.
Teoretická část bude obsahovat úvod do webových služeb a obecného testování SW. Dále návrh testovacího scénáře a porovnání testovacích nástrojů pro webové služby.
Praktická část bude obsahovat vytvoření a vystavení webové služby, uplatnění testovacího scénáře, vytvořeného v teoretické části práce a jeho následnou aplikaci na vybraném testovacím nástroji.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

PATTON, Ron. Testování softwaru. Vyd. 1. Praha: Computer Press, 2002. ISBN 80-7226-636-5
CHAPPELL, David A. a Tyler JEWELL. Java Web Services. Sebastopol, CA: O'Reilly & Associates, Inc., 2002. ISBN 0-596-00269-6
BARESI, Luciano. Test and Analysis of Web services [online]. Springer, 2007 [cit. 2013-10-05]. ISBN 978-3-540-72912-9. Dostupné z: http://link.springer.com/book/10.1007/978-3-540-72912-9/page/1

Vedoucí bakalářské práce: **Ing. Zdeněk Šilar**
Katedra informačních technologií

Datum zadání bakalářské práce: **20. prosince 2013**
Termín odevzdání bakalářské práce: **9. května 2014**

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

Ing. Lukáš Čegan, Ph.D
vedoucí katedry

V Pardubicích dne 31. března 2014

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na mou práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 2. 12. 2014 Zdeňka Repáňová

Poděkování

Mé poděkování patří Ing. Zdeňkovi Šilarovi, Ph.D. za odborné vedení, trpělivost a ochotu, kterou mi v průběhu zpracování bakalářské práce věnoval. Dále bych chtěla poděkovat všem mým pracovním kolegům, bez jejichž přičinění by finální verze práce dopadla zcela jinak. V neposlední řadě bych také ráda poděkovala svému příteli za psychickou podporu a své mamince, za to, že mě nakonec nechala prosadit si svou.

Anotace

Cílem práce je vytvoření testovacího scénáře a jeho uplatnění na webovou službu. První část práce je věnována obecnému testování softwaru a úvodu do teorie webových služeb. V další části je popsán optimální testovací scénář a jsou zde porovnány funkcionality vybraných testovacích nástrojů. Poslední část je věnována vytvoření a vystavení webové služby, grafickému rozhraní vybraného testovacího nástroje a implementaci navrženého testovacího scénáře.

Klíčová slova

testování softwaru, webová služba, WSDL, SOAP, SoapUI

Title

Web Services testing.

Annotation

The goal of this work is to create a test scenario and its application to web service. First part of work is devoted to general software testing and introduction to theory of web services. Next part describes optimal test scenario and there is comparison of selected testing tools functionality. The last part is devoted to creation and exposition of web service, graphic interface of selected testing tool and implementation of designed test scenario.

Keywords

software testing, web service, WSDL, SOAP, SoapUI

Obsah

Seznam zkratek **8**
Seznam obrázků **9**
Seznam tabulek **9**
Úvod **10**
1. Testování softwaru **11**
1.1 Význam testování při vývoji softwaru 11
1.2 Chyby v softwaru 11
1.2.1 Příčiny vzniku chyb a jejich druhy 12
1.2.2 Náklady na odstranění chyb 12
1.3 Rozdělení testů 14
1.3.1 Statické a dynamické testování 14
1.3.2 Černá a bílá skříňka 14
1.3.3 Automatické a manuální testování 14
1.3.4 Testy splnění a selhání 14
1.3.5 Progresní a regresní testy 15
1.3.6 Úroveň pokrytí 15
1.3.7 Dimenze kvality 15
1.3.8 Stadium vývoje 16
2. Dokumentace k testování **17**
2.1 Testovací plán 17
2.2 Testovací případ 17
2.3 Testovací scénář 17
3. Webové služby **18**
3.1 WSDL 19
3.2 SOAP 20
3.3 Tvorba a vystavení webové služby 20
3.3.1 Způsob Bottom-Up 20
3.3.2 Způsob Top-Down 21
3.4 Vystavení webové služby na server 22
4. Nástroje pro testování webových služeb **23**
4.1 SoapUI 24

4.2 SOAPSonar 25
4.3 Examine 27
4.4 Storm 28
5. Tvorba a aplikování testovacích scénářů 30
5.1 Testovací plán 31
5.2 Tvorba funkčních testů v testovacím nástroji 32
5.3 Vyhodnocení průběhu testování 37
5.4 Generování reportu o průběhu testování 37
5.5 Zátěžové testování 38
5.6 Řešení univerzálnosti testů pro více prostředí 40
Závěr 41
Literatura 42

Seznam zkratek

CGI	Common Gateway Interface
CORBA	Common Object Request Broker Architecture
CÚ ER	Centrální úložiště elektronických receptů
ESB	Enterprise Service Bus
FAQ	Frequently Asked Questions
GUI	Graphical User Interface
GUID	Globally unique identifier
HTTP	Hypertext Transfer Protocol
REST	Representational State Transfer
RLPO	Registr pro léčivé přípravky s omezením
SLA	Service-Level Agreement
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol, Service Oriented Architecture Protocol
URL	Uniform Resource Locator
WS	Web Service, webová služba
WSDL	Web Services Description Language
WSS	Web Services Security
XML	Extensible Markup Language
XSD	XML Schema Definition

Seznam obrázků

Obrázek 1 - Náklady na opravu chyby v průběhu vývoje softwaru [2] 13
Obrázek 2 - Vztah mezi finančními náklady a množstvím testů [2] 13
Obrázek 3 - Schematické shrnutí WSDL [11] 19
Obrázek 4 - Struktura SOAP 20
Obrázek 5 - Příklad zdrojového kódu s anotacemi 21
Obrázek 6 - Grafické znázornění WSDL 21
Obrázek 7 - Textová reprezentace WSDL 22
Obrázek 8 - GUI SoapUI 5.0 24
Obrázek 9 - GUI SOAPSonar Personal 6 26
Obrázek 10 – GUI Examine 2.5.1.-server 27
Obrázek 11 - GUI Storm r1.1-Adarna 29
Obrázek 12 - Struktura projektu 32
Obrázek 13 – Pomocný testovací krok obsahující skript pro generování unikátního identifikátoru 33
Obrázek 14 - Seznam proměnných vytvořených v rámci jednoho projektu 34
Obrázek 15 - Struktura okna pro testovací krok operace 35
Obrázek 16 - Průběh všech testů v projektu 36
Obrázek 17 - Ukázka z reportu o průběhu testování: souhrnné výsledky testů 37
Obrázek 18 - Ukázka z reportu průběhu testování: výsledky jednotlivých kroků v rámci jednoho testovacího případu 38
Obrázek 19 - Struktura testovacího případu určeného pro zátěžové testování 38
Obrázek 20 - Pracovní okno pro zátěžové testování 39
Obrázek 21 - Ukázka části skriptu pro vyplnění přihlašovacích údajů 40

Seznam tabulek

Tabulka 1 – Hodnocení funkcionalit testovacích nástrojů 23

Úvod

Proces tvorby softwaru prošel v posledních desetiletích dramatickým vývojem a dostal se do stavu, kdy jsou na něj kladeny stejné podmínky a nároky, jako na kteroukoli jinou produkční oblast. Stejně jako v jiných odvětvích jsou i zde kladeny značné požadavky na kvalitu dodávaného produktu, na rychlost jeho vzniku a ekonomickou efektivitu samotné tvorby i následné údržby.

V rámci vývoje je nutné neustále kontrolovat kvalitu vytvářeného produktu, čemuž se přizpůsobují metodiky tvorby softwaru i programové nástroje, které jsou pro kontrolu kvality využívány. Tyto nástroje se liší v návaznosti na povahu vyvíjeného softwaru – jiné nástroje jsou používány pro testování graficky orientovaných aplikací a jiné pro testování nevizuálních aplikací.

Tato práce se zabývá testováním druhé jmenované skupiny aplikací, konkrétně aplikací vystavených ve formě webových služeb. Pro tyto aplikace je typické zveřejnění abstraktního rozhraní, popisujícího očekávané vstupy a výstupy. Ty následně konzumují aplikace libovolné architektury, běžící na libovolné platformě. Přestože je možné vystavenou aplikaci testovat i s využitím reálných napojených aplikaci, je vhodnější pro testování využívat specializované nástroje.

Nástroje pro testování webových služeb neimplementují žádnou specializovanou vnitřní logiku, kterou musí implementovat reálné aplikace, a díky tomu umožňují snadné vytváření různých, často i nestandardních scénářů. Umožňují také automatizované opakované spouštění již vytvořených scénářů, stejně jako vytváření souhrnných reportů o průběhu testování apod.

Existuje řada nástrojů určených pro testování webových služeb, které se liší rozsahem svých funkcionalit, ergonomií ovládání, licenčními podmínkami použití atd. Tato práce se zabývá srovnáním některých těchto nástrojů, základy teorie testování a aplikací této teorie na nástroj, který ze srovnání vychází nejlépe.

1. Testování softwaru

Nejobecněji bývá testování považováno za samostatnou disciplínu, jejímž úkolem je ověřování kvality. Proces testování je tedy podmnožinou procesu ověřování a plánování kvality [1].

1.1 Význam testování při vývoji softwaru

Za vznikem softwaru vždy stojí nějaké potřeby. Kvalitní software je takový, který tyto potřeby plně uspokojuje. Abychom aplikaci testovali, je nutné její spuštění a existence sady podmínek a pravidel, oproti kterým bude kontrolována. Dané podmínky a pravidla vycházejí ze zmíněných potřeb pro vznik softwaru. Testování je založeno na porozumění produktu a potřebám zákazníka, pro kterého je produkt určen [1].

V počátku vývoje je nutno stanovit měřítka kvality a zvolit vhodný přístup pro jeho implementaci. Tímto se ujistíme, že už proces vývoje produktu bude dosahovat požadované kvality. Pro účely testování se poté stanoví rozsah, konkrétní testy a nástroje, které budou potřebné. V neposlední řadě dochází ke sběru dat potřebných k testování. Testování je na vstupní data citlivé, měla by tedy být realistická - generování jednotvárných dat proto není ideální [1].

Navrhované testy jsou jedinečné, a to v závislosti na tom, kdo je navrhuje. Pro jeden produkt mohou být navrženy různé sady testů s různými náklady. Pro dobré pokrytí testy je nutné správně rozdělit testy dle jejich zaměření a vybrat protikladné a doplňující techniky, které jsou snadno proveditelné a jejich aplikace je levná [1].

S příchodem náročnějších aplikací a s rostoucími požadavky na kvalitu vyvíjeného softwaru se stalo nalezení chyb obtížnějším a pracnějším, proto došlo ke vzniku řady testovacích nástrojů, které mají usnadnit a zlevnit proces testování. Takovéto nástroje mohou urychlit, částečně nahradit a automatizovat práci testera. Jde však pouze o nástroje, které nejsou schopné plně ohodnotit většinu aspektů kvality [1].

Nestačí však jen samotné nalezení chyby, je třeba se podílet i na její nápravě. K tomu nejlépe poslouží předání přehledného reportu vývojovému oddělení [1].

1.2 Chyby v softwaru

Pro určení chyby je třeba definovat pojem **specifikace produktu**. Jedná se o dohodu mezi členy vývojového týmu daného softwaru a zadavatelem. Specifikace definuje vytvářený produkt, podrobně popisuje, jak bude vypadat a jak se bude chovat. Tato dohoda může mít různou podobu, od jednoduchého ústního souhlasu až po formalizovaný, psaný dokument [2].

Pro účely softwarového průmyslu mluvíme o chybě tehdy, pokud je splněna alespoň jedna z těchto podmínek [2]:

1. Software nedělá něco, co by dle specifikace dělat měl.
2. Software dělá něco, co by dle specifikace dělat neměl.
3. Software dělá něco, o čem se specifikace nezmiňuje.
4. Software nedělá něco, o čem se specifikace sice nezmiňuje, ale měla by se zmiňovat.
5. Software není srozumitelný (nebo jen obtížně), těžko se s ním pracuje, je pomalý nebo by jej koncový uživatel nemusel považovat za správný.

1.2.1 Příčiny vzniku chyb a jejich druhy

Dle [2] nejčastější příčiny vzniku chyb u softwaru nalezneme ve specifikaci. Ta může být jak úplně vynechána, tak nedostatečně podrobná nebo se neustále měnící. V neposlední řadě může dojít k nejasnostem mezi členy vývojového týmu. Druhou hlavní příčinou vznikajících chyb je návrh softwaru. I v tomto případě jsou příčiny stejné, jako u specifikace, tj. uspěchání návrhu, časté změny návrhu nebo nedostatečně prodiskutovaný návrh mezi členy týmu. Jako další můžeme jmenovat například chyby v programovém kódu. Příčinou často bývá složitost kódu, nedostatečně podrobná dokumentace, časová tíseň apod.. Mnoho chyb v kódu vychází z chyb ve specifikaci nebo návrhu softwaru

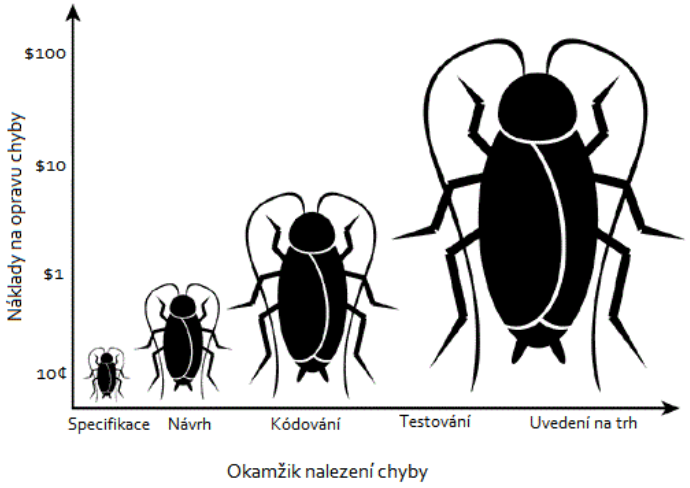
Dle [3] jsou nejčastější chyby vznikající při vývoji (tzv. softwarové). Chyba ovšem může vzniknout už na úrovni analýzy. Tento druh chyb se obvykle označuje jako chyba analytická nebo dokumentační. Reálně je podobných chyb hlášeno poměrně málo. Tento fakt nevychází ani tak z nižší chybovosti analytiků oproti vývojářům, ale je spíš dán skutečností, že testování analýzy, ze kterého by analytické chyby vzešly, není příliš rozšířené. Mnohdy se tak stává, že chyba, která je ze své podstaty analytická, je nalezena až při testování aplikace a je označena jako softwarová. Její případné překvalifikování je otázkou nastavení procesů při vývoji softwaru. Chyby aplikace mohou být způsobeny také nekorektní instalací, pak mluvíme o chybách instalace. Může jít například o nenainstalování všech součástí aplikace, nebo instalaci nesprávných verzí jednotlivých částí. Jako zvláštní typ instalačních chyb mohou být hlášeny chyby konfigurační. Jde především o nesprávné (nebo chybějící) nastavení různých parametrů a cest. Dále může být hlášena také chyba dat, která se týká především obsahu číselníků a dalších databázových tabulek, které jsou pro spuštění aplikace nezbytné.

1.2.2 Náklady na odstranění chyb

Výše nákladů na odstranění chyby roste v závislosti na fázi vývoje. Fáze zahrnuje:

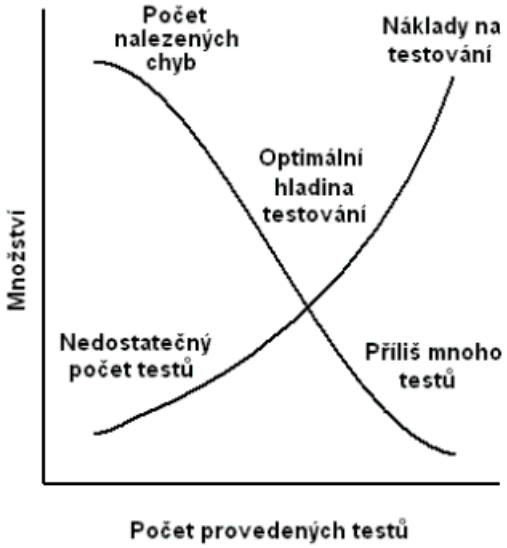
- specifikaci,
- návrh,
- kódování,
- testování,
- uvedení na trh.

Náklady na odstranění chyby, odhalené při specifikaci nebo návrhu, jsou mnohonásobně nižší než náklady na odstranění v případě, že je chyba odhalena až po uvedení softwaru na trh, viz Obrázek 1 [2].



Obrázek 1 - Náklady na opravu chyby v průběhu vývoje softwaru[1] [2]

Náklady na objevení a opravení chyb, tedy náklady na testování, jsou únosné až do meze, kdy se začnou rovnat nebo jsou vyšší, než náklady na ponechání chyby v softwaru. Je však nutné znát správné vyčíslení obou nákladů. Proto, jak ukazuje Obrázek 2, pro každý softwarový projekt a pro každou množinu testů existuje efektivní hladina testování [2].



Obrázek 2 - Vztah mezi finančními náklady a množstvím testů [2]

[1] Vyobrazení hmyzu dle anglického bug = štěnice, brouk (hmyz), ale také označení pro chybu v softwaru.

1.3 Rozdělení testů

Testy softwaru lze rozdělit do několika kategorií. Mnohdy se využívá i spojení více kategorií, jako např. „statické testování černé skříňky“.

1.3.1 Statické a dynamické testování

Toto rozdělení se dotazuje na nutnost spuštění softwaru. **Statické testování** nevyžaduje běh testovaného programu. Je tedy možné s ním začít ještě před vytvořením první verze programu. Zde můžeme zařadit například procházení zdrojového kódu.

Dynamické testování naopak probíhá pouze za běhu programu a to na základě poskytnutí vstupů a hodnocení výstupů [1, 2].

1.3.2 Černá a bílá skříňka

Při tomto rozdělení zohledňujeme znalost vnitřní struktury programu. V případě **černé skříňky** se zaměřujeme pouze na znalost vstupů a výstupů bez vědomí toho, jak je program implementován. Jinak řečeno, vidíme pouze to, jak se program chová navenek. Cílem je, co nejvíce se přiblížit skutečnému potenciálnímu uživateli. V této kategorii nalezneme například testy uživatelského rozhraní.

V případě **bílé skříňky** má naopak tester přístup ke zdrojovému kódu. Hodnotí tedy jak dění na povrchu, tak uvnitř skříňky.

V této kategorii vznikl ještě mezistupeň, tzv. **šedá skříňka**. V případě šedé skříňky má sice tester informace o vnitřní implementaci, není jich však tolik, aby bylo možno zařadit testování do kategorie bílé skříňky [1, 2].

1.3.3 Automatické a manuální testování

Pokud je testování prováděno čistě pomocí softwaru, mluvíme o **automatickém testování**. To je vhodné zejména pro opakované spouštění velkého množství testů nebo pro velké množství generovaných dat a téměř nutné pro zátěžové testování.

Pro **manuální testování** je třeba lidské ohodnocení nebo aplikace rozličných přístupů. Je tedy vhodné, aby testování bylo prováděno a vyhodnoceno člověkem [1, 2].

1.3.4 Testy splnění a selhání

Toto rozdělení se zaobírá nalezením/nenalezením cesty programem. V případě **hledání úspěchu (splnění)** je program v prvotním stádiu vývoje, je tedy pozitivním znamením, pokud nalezneme nějakou cestu k cíli a test průchodu (test-to-pass) bude úspěšný.

Naopak v případě **hledání neúspěchu (selhání)**, testování probíhá až na konci vývoje, kdy už je nalezení cesty samozřejmost, snažíme se tedy najít případy, kdy dojde k selhání. V tom případě mluvíme o testu k selhání (test-to-fail) a test je neúspěšný [2].

1.3.5 Progresní a regresní testy

Progresní testy využíváme při kontrole nových funkcí nebo vlastností aplikace. K jejich správnému provedení je nutná dokumentace, která přesně popisuje nově implementované oblasti.

Regresní testy se využívají při opětovném testování funkcí a vlastností aplikace. Jejich smyslem je ověření, že provedené změny či implementace nových vlastností v aplikaci nemělo žádný vliv na stávající funkce a vlastnosti [4].

1.3.6 Úroveň pokrytí

Testy můžeme dělit také dle toho, jakou část softwaru pokrývají [1]:

Požadavky: Kontroluje se splnění požadavků zadaných zákazníkem, pro kterého se software vyvíjí. Požadavky jsou typicky zadávány formou uživatelských scénářů, které jsou simulovány pomocí testů.

Funkce: Testují se samostatné funkční jednotky programu. Kombinací některých funkcí programu mohou být realizovány výše uvedené scénáře, ale program může obsahovat i další funkce, například podpůrné, diagnostické apod..

Zdrojový kód: Pokrytí zdrojového kódu se dále dělí na:

- **Pokrytí příkazů (řádků)** – kontroluje se, zda existuje test na daný příkaz (řádek).
- **Pokrytí hran (rozhodování)** – kontroluje se, zda v případě podmínek byly testovány kladné i záporné vyhodnocení.
- **Pokrytí podmínek** – kontroluje se vyhodnocení všech podčástí složených podmínek.
- **Pokrytí cest –** kontroluje se, zda jsou otestovány všechny možné průchody kódem.

1.3.7 Dimenze kvality

Dělení vychází z požadavků a očekávání uživatelů. Souhrnně bývají tyto dimenze označovány zkratkou FURSP+ [1]:

- **Funkčnost (Functionality) -** Kontrola správného chování funkcí systému.
- **Použitelnost (Usability)** – Kontrola uživatelské přívětivosti systému.
- **Spolehlivost (Reliability)** – Kontrola neměnného chování systému (po chybě, přetížení).
- **Podporovatelnost (Supportability)** – Kontrola snadné údržby sytému (instalace apod.).
- **Výkon (Performance)** – Kontrola výkonu systému (rychlost, připojení většího množství uživatelů).
- **Ostatní (+)** – Lokalizovatelnost (převod do jiných jazyků), kompatibilita s ostatním softwarem či hardwarem, bezpečnost atd.

1.3.8 Stadium vývoje

Testování se dělí na vícero stupňů v závislosti na úrovni, na které se testování provádí a na tom, jaký je časový horizont od napsání kódu.

Testování programátorem (Developer testing): Testování programátorem se provádí jako první a dochází k němu bezprostředně po napsání kódu. Programátor sám testuje funkčnost svých metod. Dochází zde ke kontrole funkčnosti kódu a kontrole toho, zda kód dělá to, co má. Odhalení chyby v tomto kroku vede k nejmenším nákladům na opravu [1].

Testování jednotek (Unit testing): Jedná se o proces testování co možná nejmenších částí kódu (u objektově orientovaného programování jsou to metody a třídy). Pro usnadnění a úsporu času se používají nástroje na bázi frameworků[2]. Jedná se o testy automatické, opakovatelné [1].

Funkční testování (FAT, Factory acceptance tests): Funkční testy ověřují, že aplikace správně plní všechny úkoly, pro které je dle požadavků zákazníka určena. Do této kategorie patří i tzv. **nefunkční testování**, které spočívá v testování všech vlastností aplikace, které přímo nesouvisí s jejími funkcemi, ale zároveň jsou podstatné pro její správné fungování [5].

Integrační testování (Integration testing): Integrační testování se zaměřuje na funkčnost integrace subsystémů. Kontroluje se zde správné chování nového podsystému vůči zbytku systému [1].

Systémové testování (System testing): Během systémových testů dochází k otestování aplikace jako celku. Probíhá hlavně v týmech, kde na jednotlivých funkčnostech pracují samostatné skupiny. Součástí této fáze jsou např. testy výkonnosti, spolehlivosti nebo bezpečnosti [1].

Akceptační testování (UAT, User acceptance testing): K akceptačním testům dochází při předávání produktu zákazníkovi. Testy často běží dle připravených scénářů, předem dohodnutých oběma stranami [1].

Pilotní testování (Pilot testing): Zkušební provoz v reálném provozu, tedy ověření správné funkce i při napojení na reálné okolí, na rozdíl od simulovaných prostředí v laboratořích [6].

[2] Softwarová struktura, která slouží jako podpora při programování, vývoji a organizaci jiných softwarových projektů.

2. Dokumentace k testování

Vytváření dokumentace k testování umožnuje testovacímu týmu především [7]:

- vytvoření konkrétních scénářů pro testy,
- definování požadavků na testovací data,
- určení požadavků na podporu ze strany dalších pracovních týmů,
- zadání požadavků na technické vybavení.

2.1 Testovací plán

Testovací plán stanovuje základní podmínky, za kterých bude testování probíhat, a obsahuje tyto informace [7]:

- **Cíl testovaní** – definuje, čeho by mělo být testováním dosaženo (ověření všech funkcí, dodatečná kontrola přidaných komponent apod.).
- **Přehled plánovaných testů** – návrh takového seznamu testů, který postihne všechny funkčnosti aplikace, s důrazem na použití ze strany zákazníka.
- **Stanovení priorit** – nejvyšší prioritu mají prvky, které jsou z hlediska funkčnosti aplikace nejvýznamnější anebo ty, u kterých v případě nefunkčnosti hrozí nejvyšší riziko. Ostatní prvky jsou otestovány až po otestování prvků s nejvyšší prioritou.
- **Testovací strategie** – popisuje, jaké typy testů budou provedeny (funkční, zátěžové apod.) a jednotlivě jsou pak zařazeny do konkrétních fází testování (integrační, akceptační aj.).
- **Požadavky na zdroje** – slouží k definování požadavků, které musí být splněny, aby bylo možné testy provést (testovací hardware, testovací software, lidské zdroje).
- **Definice rizik** – slouží k vymezení situací, které mohou ohrozit úspěšnost testování (nedodání aplikace v termínu, nedostupný testovací nástroj, nedostatečné školení aj.).

2.2 Testovací případ

Testovací případ je podkladem pro testování jednoho konkrétního místa v aplikaci pro konkrétní situaci. Definuje výchozí podmínky, které musí být splněny, aby bylo možné daný případ testovat, a také jednoznačně určuje podobu očekávaného výsledku [7].

2.3 Testovací scénář

Testovací scénář vzniká spojením několika **testovacích kroků**. Snahou je simulovat konkrétní způsob použití aplikace. Obvykle se jedná o simulování procesu, jakým bude aplikace používána zákazníkem. Testovací kroky jsou ve scénáři řazeny za sebe tak, aby výstup jednoho kroku byl vstupem dalšího kroku [7].

3. Webové služby

Jedním z trendů současné aplikované informatiky je integrace různorodých, často již existujících aplikací (využití již hotové logiky) do vyšších spolupracujících celků. Ideální formou integrace je, pokud nejsou jednotlivé aplikace příliš pevně svázány a pokud každá aplikace zodpovídá za relativně malou oblast. S ohledem na různorodost stávajících informačních systémů (operačních systémů, programovacích jazyků, ve kterých jsou aplikace vyvíjeny aj.) je rovněž ideální, pokud je integrace aplikací realizovaná na platformě nezávislým způsobem. Tyto, a další požadavky, splňují velmi dobře tzv. webové služby (dále jen WS).

WS lze chápat jako ucelené části aplikační logiky, umístěné v síťovém prostředí (internet, intranet, …) a dostupné přes standardní internetové protokoly, jako je např. HTTP [8].

WS jsou založeny na standardizovaném formátu XML. Jedná se o otevřený, na platformě nezávislý jazyk. S ohledem na heterogennost současného internetu i díky tomu WS předčily dřívější technologie, jako jsou např. CORBA či CGI, které využívaly proprietární binární formáty [8].

Webové služby umožňují nadefinovat abstraktní rozhraní, prostřednictvím kterého spolu jednotlivé aplikace (komponenty) komunikují. Díky již zmíněnému neutrálnímu XML formátu a všeobecně podporovanému HTTP protokolu nemusí jednotlivé aplikace znát implementační detaily ostatních aplikací. Aplikace implementující definované rozhraní se může také kdykoli změnit bez ovlivnění ostatních aplikací [9].

Díky tomu je možné i složité procesy rozdělit do menších celků, které je možné řešit samostatně, s nižšími náklady na vývoj i následnou údržbu. Prostřednictvím webových služeb je tak možné realizovat tzv. SOA[3].

Webové služby jsou založeny na dvou základních technologiích:

1. WSDL, popisující veřejné rozhraní WS,
2. komunikačním protokolu SOAP.

Existují i jiné způsoby pro komunikaci s webovou službou. Je jím například způsob komunikace pomocí protokolu REST, který je na rozdíl od známějšího SOAP orientován datově, nikoli procedurálně. Rozhraní REST je použitelné pro jednotný a snadný přístup ke zdrojům. Zdrojem mohou být jak data, tak stavy aplikace (pokud je lze popsat konkrétními daty) [10].

Práce se však nadále bude zabývat výhradně SOAP komunikací.

[3] Architektura orientovaná na služby.

3.1 WSDL

WSDL je jazyk určený pro abstraktní popis rozhraní WS. Lze pomocí něj definovat, jaké operace daná služba umožňuje, jaké informace a v jaké struktuře jsou očekávány na vstupu a výstupu jednotlivých operací, a pomocí jakých protokolů komunikace probíhá [9].

WSDL tedy popisuje rozhraní pouze syntakticky – shrnuje názvy operací (funkcí, metod), jména a typy parametrů návratových hodnot. Lze jej tedy přirovnat např. k rozhraní v jazyku Java [11].

Jak je vidět na Obrázku 3,WSDL dokument se skládá z následujících částí [11]:

1. Definice datových typů, zejména s využitím jazyka XSD.
2. Definice zpráv využívající definované datové typy.
3. Definice rozhraní v rámci elementu portType ve WSDL 1.1 (ve WSDL 2.0 je tento element nazýván výstižněji jako interface); portType element obsahuje jeden a více logicky souvisejících elementů operation, které popisují jednotlivé funkce (název, vstupy, výstupy) s využitím dříve definovaných zpráv.
4. Definice komunikačních protokolů, pomocí kterých je možné službu volat (element binding). Ve WSDL verze 1.1 je standardizován pouze HTTP protokol, tzn., že na jednoznačném označení případných jiných protokolů se musí komunikující strany předem dohodnout.
5. Lokalizace služby.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="PrvniSluzba"  targetNamespace="urn:mojeURI"
 xmlns="http://schemas.xmlsoap.org/wsdl/" ...  >
<types>
  ... definice datových typů ....
</types>

<message>
 ... definice komunikačních zpráv pomocí typů ...
</message>

<portType>
 ... definice operací pomocí komunikačních zpráv ...
</portType>

<binding> ... že se volá přes HTTP ... </binding>
<service> ... na jakém URL (stroji, portu) se volá ... </service>

</definitions>
```

Obrázek 3 - Schematické shrnutí WSDL [11]

Protože se jedná o dokument ve formátu XML, je snadno strojově zpracovatelný, čehož se využívá zejména pro různé generátory zástupných kódů v jednotlivých programovacích jazycích, ale i pro různé vizualizace apod. [11].

3.2 SOAP

SOAP je protokol pro výměnu zpráv. Původně vznikl jako protokol, který popisoval způsob přenosu XML dokumentů prostřednictvím metody POST protokolu HTTP. S postupným vývojem byla myšlenka přenosu XML dokumentů rozšířena i na další přenosové protokoly (např. SMTP), přesto je protokol HTTP stále nejběžnějším [9].

SOAP je de facto implementací XML návrhového vzoru obálka (envelope). Struktura vychází z kořenového elementu Envelope, který obsahuje nepovinný element Head a povinný element Body. Zasílaný XML dokument je uveden v elementu Body a jeho struktura již není standardem SOAP blíže specifikována (sekundárně ale vyplývá např. z popisu příslušné operace v definici WSDL). Nepovinná hlavička obsahuje různé „metainformace“, jako jsou například autentizační údaje, transakční data, kontextové informace apod. Často jsou tyto doplňující informace specifikovány dodatečnými standardy, jako například WSS apod. [9, 12].

Následující Obrázek 4 ilustruje strukturu SOAP. Kromě základní struktury je z ní také patrné intenzivní využívání jmenných prostorů XML.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" >
   <SOAP-ENV:Body>
      <m:AppPingDotaz xmlns:m=http://www.sukl.cz/rlpo/v100
xmlns:m0="http://www.sukl.cz/dt/v230">
         <m:ID_Zpravy>CC498F40-D36A-4377-97F8-845C45B9D382</m:ID_Zpravy>
         <m:VerzeRozhrani>1.00</m:VerzeRozhrani>
         <m:DatumCasOdeslaniZpravy>2001-12-17T09:30:47Z</m:DatumCasOdeslaniZpravy>
         <m:SW_Klienta>
            <m0:Vyrobce>NazevFirmy</m0:Vyrobce>
            <m0:Nazev>NazevSW</m0:Nazev>
            <m0:Verze>1.0</m0:Verze>
         </m:SW_Klienta>
      </m:AppPingDotaz>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Obrázek 4 - Struktura SOAP

3.3 Tvorba a vystavení webové služby

Webovou službu je možné vystavit dvěma způsoby:

3.3.1 Způsob Bottom-Up

Tento způsob umožňuje vytvoření WS ze zdrojového kódu opatřeného příslušnými anotacemi (viz Obrázek 5). I přesto, že se tento způsob vystavení webové služby na první pohled jeví jako jednodušší, u rozsáhlejších webových služeb se začínají objevovat určité nevýhody. Například, že tvorba WSDL vzniká až kompilaci zdrojového kódu, nebo že výsledná XSD/XML bývají hůře čitelná pro běžného uživatele. V případě výskytu chyby v generátoru výsledného WSDL, je oprava velmi zdlouhavá.

```java
package cz.sample;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;

@WebService
public class SampleWS implements SampleWSLocal {

    public SampleWS() {
    }

    @WebMethod
    public Book getBook(@WebParam(name = "bookId") String bookId) {
        return Book.get(userId);
    }

}
```

Obrázek 5 - Příklad zdrojového kódu s anotacemi

3.3.2 Způsob Top-Down

Tento způsob vytváří zdrojový kód z WSDL. Příslušný framework vygeneruje třídy, které opatří anotacemi. Pro vytváření WSDL existuje řada programů, z nichž nejznámější je pravděpodobně Altova XMLSpy. Pomocí tohoto nástroje je možné vytvořit WSDL pomocí grafického editoru (viz Obrázek 6) i zobrazit v textové podobě (viz Obrázek 7).

Tento způsob byl použit při tvorbě WS, na které budou demonstrovány testovací scénáře v další kapitole.

Obrázek 6 - Grafické znázornění WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:description targetNamespace="http://new.webservice.namespace" xmlns:wsdl="http://www.w3.org/ns/wsdl"
xmlns:wsoap="http://www.w3.org/ns/wsdl/soap" xmlns:whttp="http://www.w3.org/ns/wsdl/http"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://new.webservice.namespace">
    <wsdl:types>
        <xs:schema targetNamespace="http://new.webservice.namespace" elementFormDefault="qualified">
            <xs:element name="BookRequest" type="xs:string"/>
            <xs:element name="BookResponse" type="xs:string"/>
        </xs:schema>
    </wsdl:types>
    <wsdl:interface name="SampleWS">
        <wsdl:operation name="getBook" pattern="http://www.w3.org/ns/wsdl/in-out">
            <wsdl:input messageLabel="In" element="tns:BookRequest"/>
            <wsdl:output messageLabel="Out" element="tns:BookResponse"/>
        </wsdl:operation>
    </wsdl:interface>
    <wsdl:binding name="SampleWS" interface="tns:SampleWS" type="http://www.w3.org/ns/wsdl/soap"
    wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/">
        <wsdl:operation ref="tns:getBook" wsoap:mep="http://www.w3.org/2003/05/soap/mep/request-response/"/>
    </wsdl:binding>
    <wsdl:service name="SampleWS" interface="tns:SampleWS">
        <wsdl:endpoint name="NewEndpoint" binding="tns:SampleWS"/>
    </wsdl:service>
</wsdl:description>
```

Obrázek 7 - Textová reprezentace WSDL

Pro oba způsoby existují k různým vývojovým prostředím různé typy frameworků.

3.4 Vystavení webové služby na server

Vystavení WS je principiálně stejné, jako vystavení jakékoli jiné webové aplikace. Nejprve je nutné provést finální kompilaci zdrojových kódů, konfigurací a ostatních souborů nezbytných pro správné spuštění a funkčnost WS. Struktura výsledného souboru se může lišit dle typu kontejneru, který bude mít na starost životní cyklus WS (inicializaci, spuštění, zastavení, restart). Pro úspěšné vystavení/konfiguraci je nutné nastudovat dokumentaci k příslušnému aplikačnímu serveru.

4. Nástroje pro testování webových služeb

Testování WS se zaměřuje především na funkčnosti, které nejsou primárně přístupné prostřednictvím GUI, a proto je vhodné testovat přímým oslovováním konkrétní WS. Ověřuje se, zda se na příslušný dotaz (request) vrací odpovídající odpověď (response). Toto nám umožňují testovací nástroje uzpůsobené pro WS [13].

Pro účely porovnávání byl zvolen jako výchozí program SoapUI a dohledány obdobné nástroje na webových stránkách alternativeTo[4]. Hodnocení jednotlivých nástrojů je shrnuto v Tabulce 1. Samotný testovací scénář, jeho provedení a vygenerování reportů bude provedeno v programu SoapUI Pro (komerční verze).

V souvislosti s testovacími nástroji a jejich vlastnostmi se nabízí přiblížení pojmu **Mock Services** neboli **Web Services Mocking**. Jedná se o typ testování, které je aplikováno v případě, že již existuje WSDL, ale stále ještě není naprogramována vnitřní logika služby. Testování pak probíhá na principu uživatelsky vyplněných vzorových odpovědí, díky kterým je umožněno předpřipravení a propojení testovacích kroků a scénářů. Až je vnitřní logika naprogramována, přenastaví se koncové body a na dotazy začnou chodit odpovědi s vlastní vnitřní logikou.

Tabulka 1 – Hodnocení funkcionalit testovacích nástrojů

	SoapUI	SOAPSonar	Examine	Storm
Funkční testování	3	3	3	2
Zátěžové testování	3	0	0	0
Mock Services	3	0	3	0
Podpora více WSDL v jednom projektu	3	0	0	3
Tvorba proměnných	3	1	0	0
Tvorba podmínek	3	0	2	0
Tvorba skriptů	3	1	0	0
Funkce pro generování dat (GUID…)	0	2	0	0
Hlavičkové soubory	2	3	0	0
Komentáře	2	2	0	0
Zabezpečení	3	3	3	1
Šifrování a elektronické podepisování	1	1	3	0
Připojení k externím zdrojům dat	3	2	0	0
Generování reportů	2	3	1	0
Zákaznická podpora	3	2	3	0
Pracovní skupiny	3	2	0	0
Relativní adresace	3	0	0	3
Přehlednost GUI	3	2	2	3
Formulářové rozhraní	0	3	3	3
Načtení ze souboru i URL	3	0	3	0
Volba jazyka	0	0	0	0

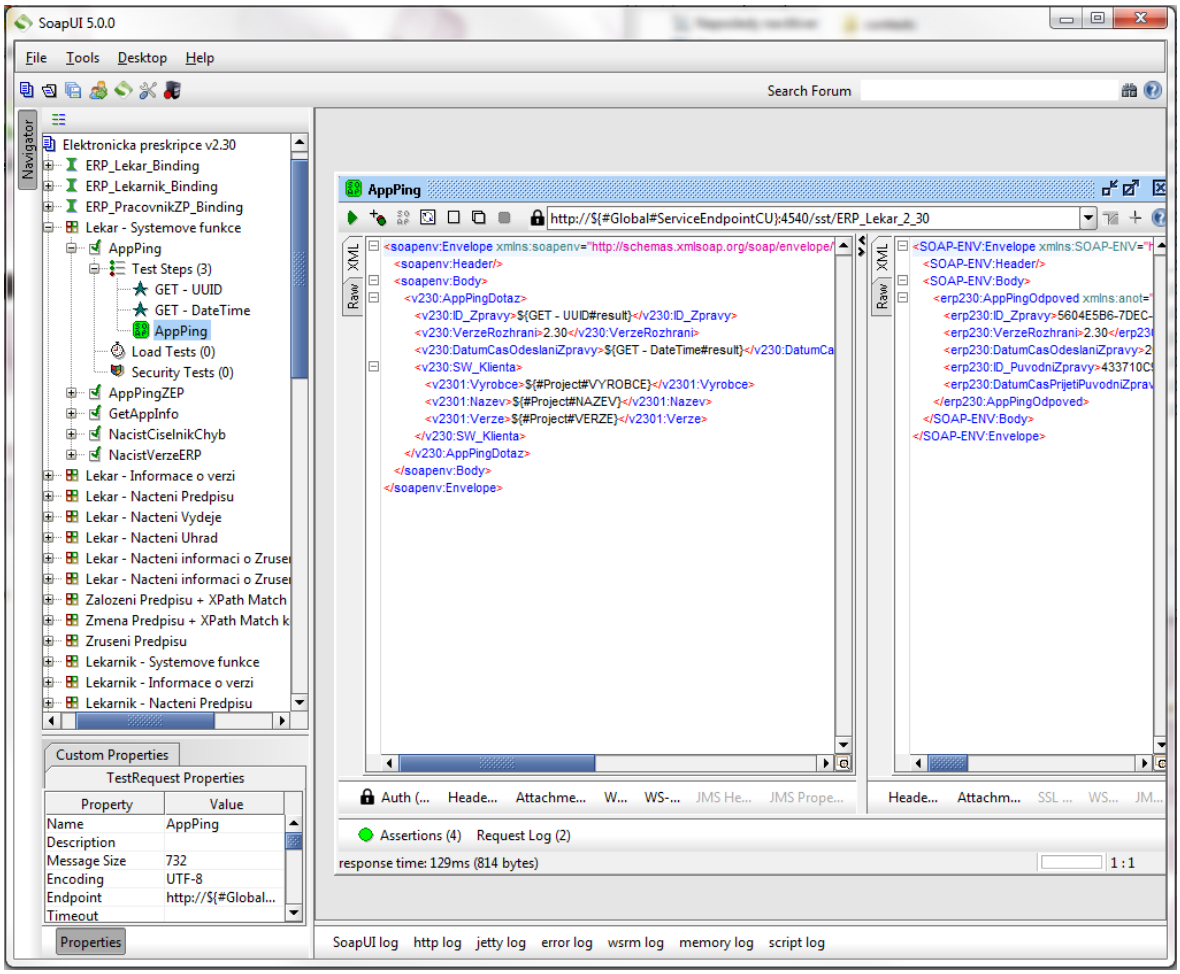
0 – funkcionalita není podporována nebo nebyla nalezena
1 – funkcionalita má minimální (omezenou) podporu
2 – funkcionalita má průměrnou podporu
3 – funkcionalita má rozšířenou podporu

[4] www.alterntiveto.net

4.1 SoapUI[5]

Jedná se o přední software pro testování SOAP a REST WS. Je produkován společností SmartBear Software. V současné době je nejnovější dostupná open source verze SoapUI 5.0 a komerční verze SoapUI Pro 5.1.2. Plná verze má oproti bezplatné rozsáhlejší zákaznickou podporu, možnost připojení k externím datovým zdrojům a podporu testovacích reportů [14].

Pro účely porovnání byla použita nezpoplatněná verze SoapUI 5.1, viz Obrázek 8.



Obrázek 8 - GUI SoapUI 5.0

Výhody:

- Podporuje Mock Services.
- Umožňuje tvorbu pracovních ploch (workspaces), které seskupují projekty do uživatelsky zvolených skupin. Po vybrání pracovní plochy se automaticky načtou všechny projekty, které jsou v této ploše přidané.

[5] http://www.soapui.org/

- Podpora uživatelských proměnných z properties souborů i proměnných vytvořených přímo v programu, a to na úrovni globální, projektové i balíčkové.
- Validace odpovědi dle vlastních podmínek (obsah a struktura XML, HTTP kódy, validace SOAP atd.).
- Možnost tvorby vlastních skriptů na více úrovních – load script (provede se při načítání projektu), setup script (provádí se nad balíčkem), assertion script (kontrolní skript pro validaci u dotazu/odpovědi) v jazyce Groovy[6].
- Plná verze umožňuje jednoduché načítání a zápis dat do externích zdrojů (databáze, Excel apod.).
- Plně podporuje relativní adresaci.
- Přehledné webové stránky obsahující tutoriály, tipy a návody pro práci programem.
- Software se neustále vyvíjí a vylepšuje.

Nevýhody:

- Chybí funkce pro generování dat, a proto je potřeba psát si vlastní skripty.
- Umožňuje přidání hlavičkových souborů pouze na úrovni dotazu.
- Bezplatná verze postrádá přehledné formulářové rozhraní - pracuje se v ní výhradně v XML.
- Bezplatná verze postrádá možnost připojení k externím datovým zdrojům pomocí testových kroků, vše zde musí být řešeno přes skripty.

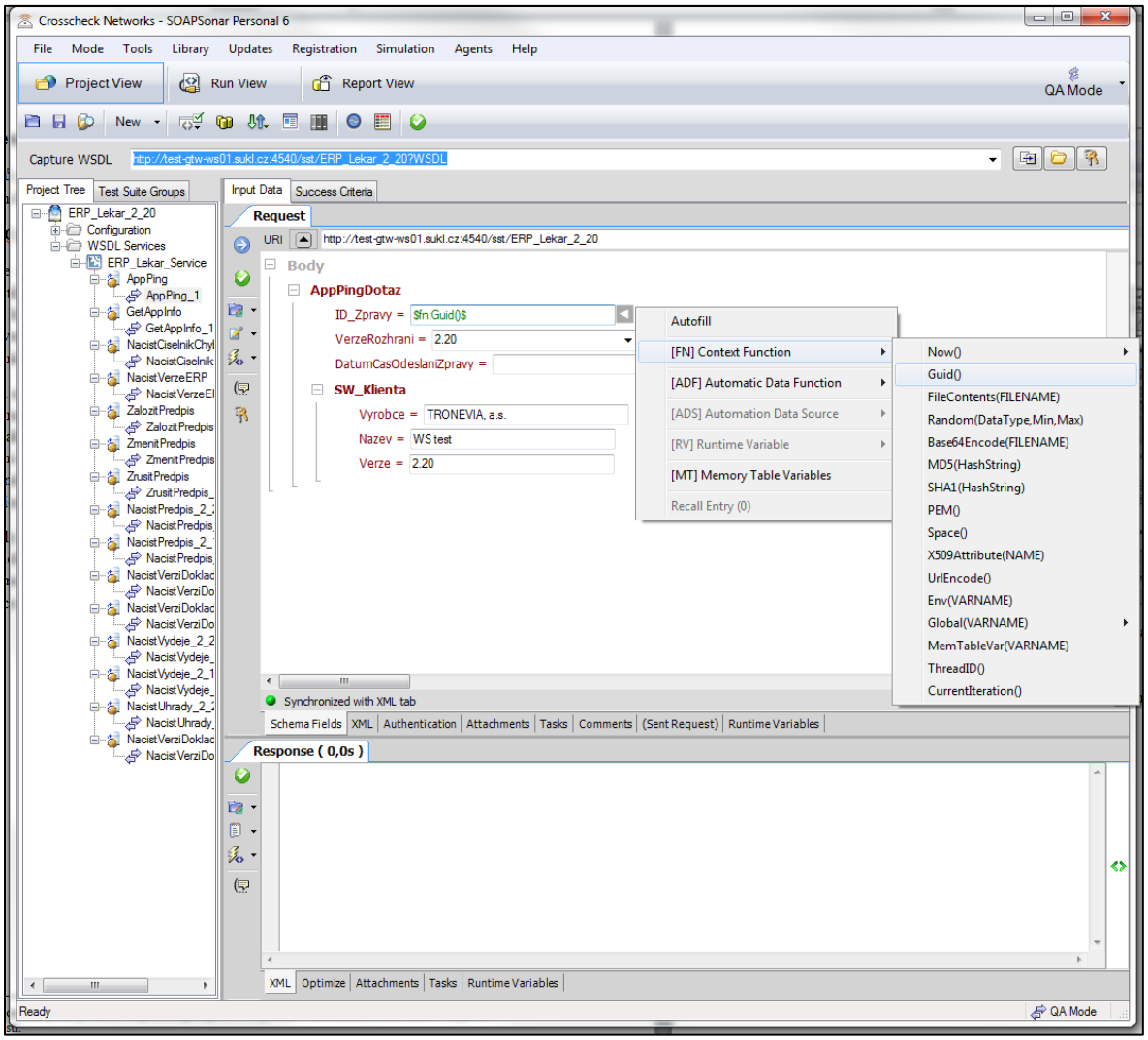
4.2 SOAPSonar[7]

Jedná se o software korporace Crosscheck Networks, určený především pro testování a diagnostiku SOAP, XML a REST služeb.

Jedinou nezpoplatněnou edicí je Personal, která nabízí možnost pouze základní konfigurace a testování. Mezi zpoplatněné verze patří Professional, která má oproti základní verzi lepší podporu tvorby vlastních proměnných, verzování WSDL, testování zabezpečení, podporu souběžných virtuálních klientů a další. Druhou zpoplatněnou edicí je Server, která nabízí kompletní sadu funkcí verzí Personal i Professional, s rozšířením pro ESB[8] nebo efektivnější testování výkonu [15].

Pro účely porovnání byla stažena základní verze Personal 6, viz Obrázek 9.

[6] Objektově orientovaný (skriptovací) programovací jazyk pro platformu Java.
[7] http://www.crosschecknet.com/products/soapsonar.php
[8] Softwarová architektura pro návrh a realizaci komunikace mezi vzájemně se ovlivňujícími aplikacemi v SOA.



Obrázek 9 - GUI SOAPSonar Personal 6

Výhody:

- Přehledné formulářové zobrazení struktury XML dotazu již v nezpoplatněné verzi.
- Nabídka hotových funkcí (generování GUID, vyplnění data v různých formátech, generování různých datových typů s parametry od-do atd.).
- Jednoduché formulářové připojení k databázi.
- Možnost přidání hlavičkových souborů na globální i nižší úrovni.
- Rozšířená možnost autentizace (Kerberos, základní HTTP, SSL).
- Umožnuje vytvoření vlastních globálních proměnných, i jejich načtení ze souboru typu properties.
- Online zákaznická podpora.

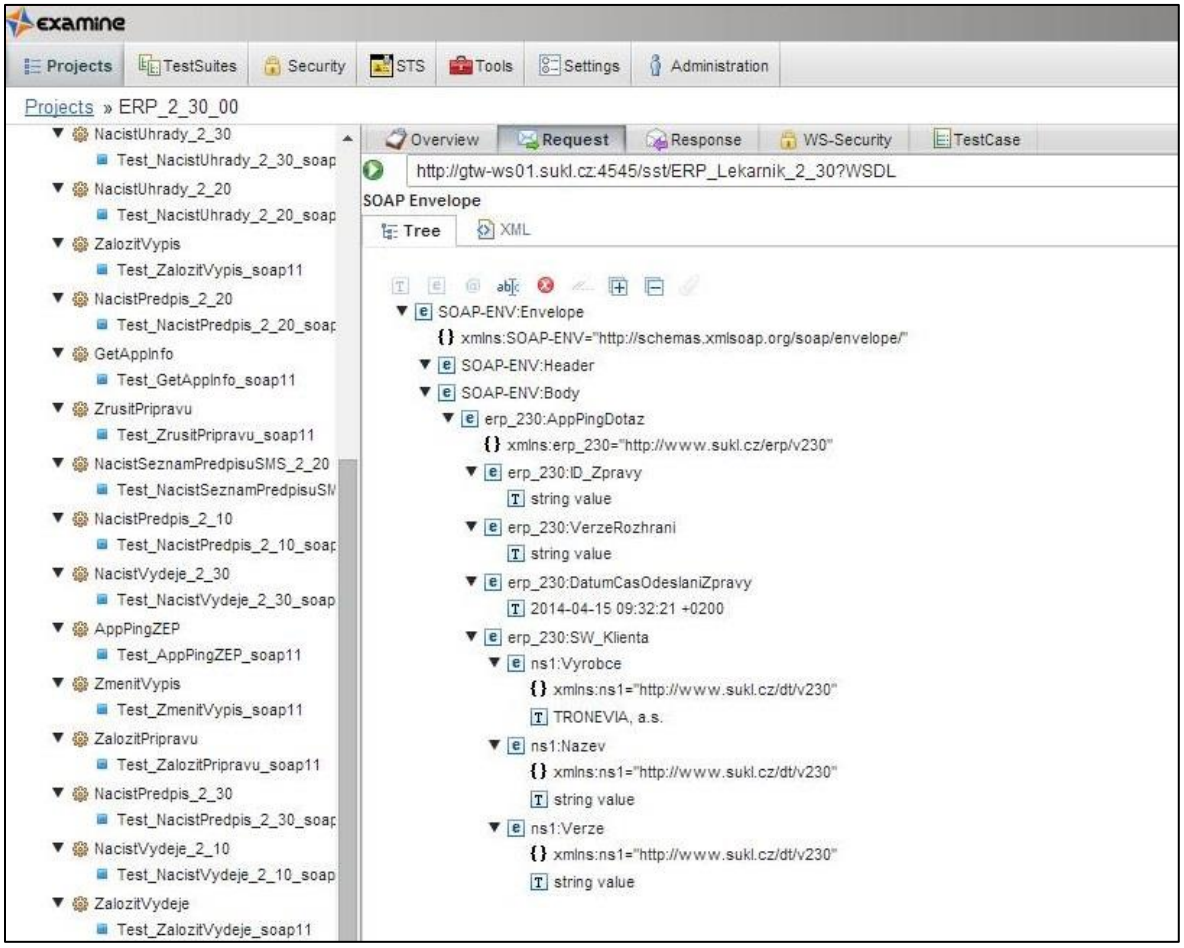
Nevýhody:

- V nezpoplatněné verzi není možnost zátěžového testování.
- Nepodporuje Mock Services.

- Problém s relativní adresací. Všechny soubory musí být nahrány v jedné složce anebo musí být dohledány a nahrány ručně.
- Chybí možnost vytvořit si pracovní skupiny, nedovoluje otevřít více projektů najednou anebo přidat více WSDL do jednoho projektu.
- Není možné vytvářet vlastní pomocné skripty.
- Neumožnuje ověřit správnost odpovědi dle vlastních zadaných pravidel (např. kontrola obsahu na výskyt řetězců apod.).
- Umožňuje načtení služby pouze ze souboru, nikoli z URL.

4.3 Examine[9]

Jedná se o komerční testovací nástroj pro WS, poskytovaný korporací Stratumsoft Technologies. Pro účely porovnání byla stažena verze 2.5.1.-server, viz Obrázek 10.



Obrázek 10 – GUI Examine 2.5.1.-server

[9] http://www.stratumsoft.com/

Výhody:

- Funguje jako webová aplikace na serveru, díky čemuž umožnuje více uživatelům současné přihlášení a používání aplikace (počet uživatelů je omezen typem zakoupené licence).
- Podporuje Mock Services.
- Umožnuje přidání dalších podmínek pro validaci odpovědi.
- Podporuje přidání elektronického podpisu.
- Podpora pro zákazníky (helpdesk, FAQ).
- Zobrazení dotazů v XML i ve formulářové podobě.
- Umožňuje načtení služby ze souboru i URL.

Nevýhody:

- Dostupná je pouze komerční verze, která je bezplatně použitelná po dobu 14 dnů.
- Nepodporuje relativní adresaci a ani ruční nahrání souborů, které nemůže nalézt.
- Neumožňuje zátěžové testování.
- Není možné otevřít více projektů/rozhraní najednou.
- K zaznamenání jakýchkoli změn je třeba projekt ukládat (např. po přenastavení koncového bodu, přihlašovacích údajů apod.).
- Neumožnuje nahrát vlastní proměnné, hlavičkové soubory nebo skripty.
- Neumožnuje připojení k externím zdrojům dat.

4.4 Storm[10]

Jedná se o open source program určený pro testování WS. Pro účely testování byla stažena verze r1.1-Adarna, viz Obrázek 11.

Výhody:

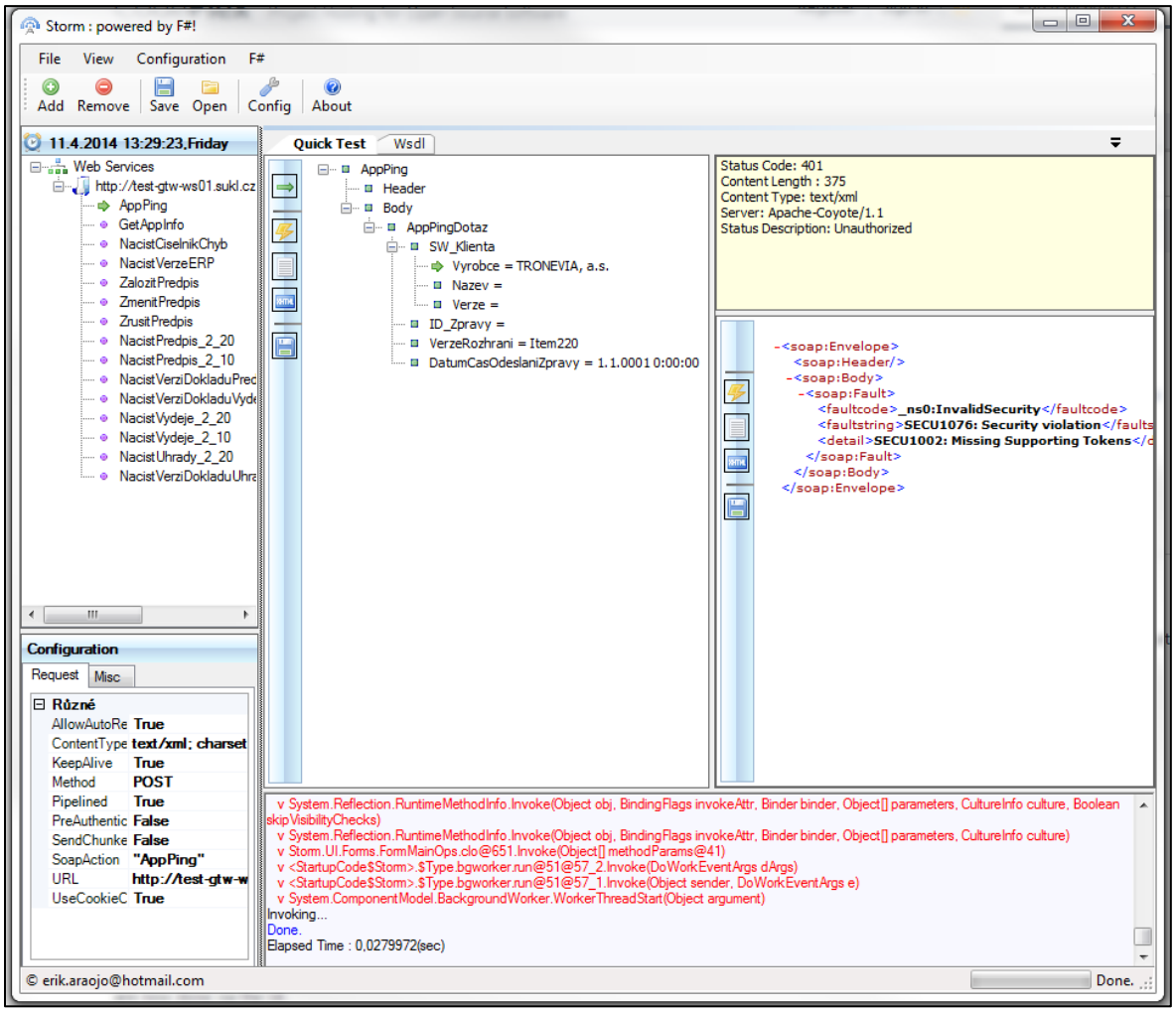
- Grafické rozhraní je přehledné a rychle reaguje.
- Formulářové zobrazení struktury dotazů.
- Podporuje relativní adresaci.
- Umožňuje otevření více projektů najednou.

Nevýhody:

- Chybí dokumentace k programu, sekce Help i další podpora pro uživatele.
- Neumožňuje provádět zátěžové testování.
- Nepodporuje Mock Services.
- Nelze vytvořit vlastní proměnné, skripty, testovací podmínky ani přidat hlavičkové soubory anebo se připojit k externímu zdroji dat.

[10] https://storm.codeplex.com

- Kromě vyplnění uživatelského jména a hesla pro jednotlivé dotazy nepodporuje žádnou další formu zabezpečení.
- Umožňuje načtení služby pouze z URL, nikoli ze souboru.
- Testy nejsou automatizované, je třeba spouštět ručně jeden dotaz po druhém.



Obrázek 11 - GUI Storm r1.1-Adarna

5. Tvorba a aplikování testovacích scénářů

Testování bude aplikováno na WS ERP verze 2.30. Zkratka ERP označuje systém zahrnující především elektronickou preskripci, výdej na elektronický recept a úhrady výdejů. Jedná se o systém, který umožňuje lékaři vystavení receptu v elektronické podobě, popř. dříve vystavený recept načíst za účelem kontroly, zda si pacient předepsaný přípravek již vyzvedl. Lékárníkovi je umožněno elektronický recept načíst a zaevidovat přípravky, které byly na základě receptu vydány. Zdravotní pojišťovně je umožněno za účelem kontroly elektronický recept načíst, event. k němu uvést výši přiznané úhrady.

Data elektronických receptů jsou uložena v tzv. Centrálním úložišti elektronických receptů (CÚ ER).

Služba obsahuje 3 základní rozhraní:

1. **Rozhraní pro lékaře**, obsahující především funkce pro práci s elektronickými předpisy (zakládání, změna, zrušení).
2. **Rozhraní pro lékárníka**, obsahující především funkce pro práci s výdeji na elektronické předpisy (zakládání, změna, úhrada, příprava, výdej z přípravy).
3. **Rozhraní pro pracovníka zdravotní pojišťovny**, obsahující především funkce pro práci s úhradami výdejů na elektronický recept (založení, změna, zrušení).

Každé z těchto tři rozhraní dále obsahuje systémové funkce sloužící pro kontrolu nastavení systému (elektronický podpis) a spojení (SSL, VPN, autentizace).

Služba se testuje na testovacím i na produkčním prostředí. Testovací prostředí slouží např. vývojářům třetích stran, kteří napojují své aplikace na zmíněnou WS. S produkčním prostředím komunikuje software koncového zákazníka. Služba na obou prostředích je funkčně identická, liší se pouze URL a sada přihlašovacích údajů (s produkčním prostředím není možné komunikovat s využitím testovacích autentizačních údajů a naopak).

V rámci WS ERP je podporován souběžný provoz více verzí služby, což umožnuje nekomplikovaný a pozvolný přechod na novou verzi. Na druhou stranu to s sebou přináší zvýšené nároky na testování, protože součástí testů musí být i ověření komunikace se staršími verzemi ERP.

Integrační testování musí zahrnovat i komunikaci ERP s dalšími systémy, proto bude ověřena také komunikace s Registrem pro léčivé přípravky s omezením (RLPO). RLPO je systém, který umožňuje jiným systémům (tedy i CÚ ER) ověřit, zda jsou splněny podmínky omezení kladené na předepsání či výdej zvláštní skupiny léčivých přípravků označovaných jako léčivé přípravky s omezením.

Testy budou kompletovány v programu SoapUI Pro 4.5.1, kde budou následně provedeny i zátěžové testy, které budou prováděny současně na několika zařízeních.

5.1 Testovací plán

Veškeré testování bude prováděno jako dynamické. Počáteční manuální testování má za cíl vytvoření automatických testů s takovými scénáři, které budou v budoucnosti pravidelně kontrolovat kvalitu aplikace.

Podmínky pro úspěšnou tvorbu testů:

- Znalost URL přístupových bodů, na kterých je vystavená WS pro produkční a testovací prostředí (VPN i SSL).
- Existence naprogramované aplikační logiky. Tato podmínka nemusí být splněna v případě, že se chystáme na testování typu Mock Services.
- Vytvoření funkčního přístupu přes SSL i VPN připojení.
- Vlastnictví funkčních přihlašovacích údajů.
- Vlastnictví technických prostředků pro vytváření zaručeného elektronického podpisu, kterým musí být zasílané zprávy opatřeny.
- Obeznámení testera s požadavky na funkčnost WS.

Při tvorbě testovacího scénáře je kladen důraz na návrh takového seznamu testů, který postihne všechny funkčnosti aplikace, především s důrazem na použití ze strany zákazníka. V tomto scénáři nejsou zahrnuty testy, které není možné provádět ve vybraném testovacím nástroji (např. testování programátorem, pilotní testování):

1. **Funkční testy**
   - **Testy splnění:** V první fázi budou otestovány všechny funkce dostupné pro jednotlivá rozhraní, jako je zakládání, změna, načítání, systémové funkce apod. Do dotazů budou vkládána validní data a cílem bude vrácení kladné odpovědi.
   - **Testy selhání**: V tomto kroku budou dotazům předkládána nestandardní data, která budou mít za následek vyvolání chyby. Všechny tyto chyby musí být ošetřeny a je potřeba na ně vracet programátorem vytvořené chybové odpovědi, které klientovi poskytnou dostatek informací k odhalení chybného vstupu.
   - **Integrační testy** budou součástí již testů splnění a selhání, protože je zde nutná vazba např. výdeje na předpis, případně úhrady na výdej (kontrola komunikace mezi jednotlivými rozhraními) nebo ověření funkcí pro načtení dokladů založených v předchozích verzích (kontroluje komunikaci mezi verzemi) apod.
2. **Nefunkční testy**
   - **Testy zabezpečení:** Testování přístupu přes SSL kanál i VPN router a ověření funkčnosti přihlašovacích údajů. Součástí jsou dále testy autentizace a autorizace.
   - **Zátěžové testy:** Zátěžové testy budou simulovat vícenásobný přístup většího počtu klientů po zvolenou dobu. Cílem je ověřit, že dostupnost a výkon služby bude odpovídat parametrům definovaným v příslušné SLA[11] smlouvě.

[11] Smlouva sjednaná mezi poskytovatelem služby a jejím konzumentem.

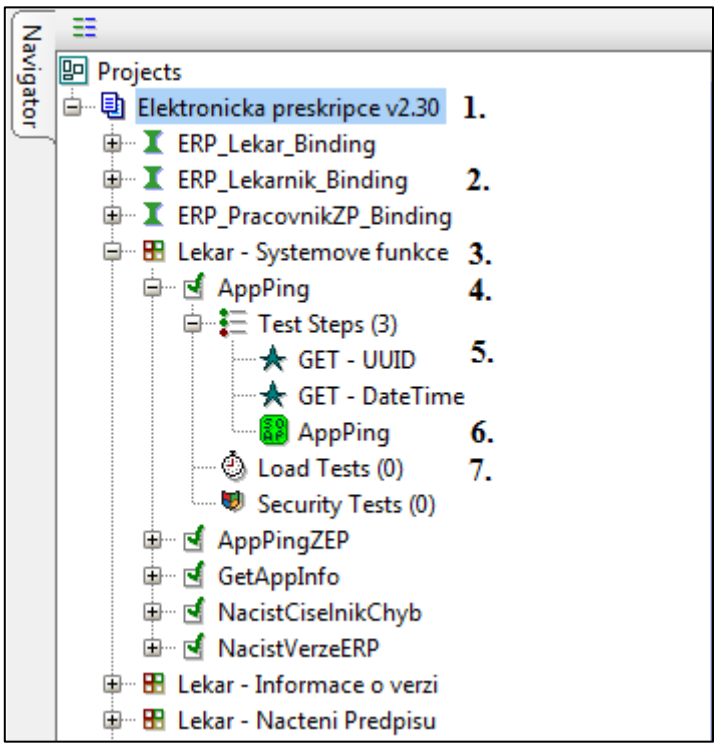
5.2 Tvorba funkčních testů v testovacím nástroji

Jak již bylo zmíněno, tvorba testů bude probíhat v testovacím nástroji SoapUI Pro 4.5.1. Problematika vytváření testů zde bude diskutována pouze ve stručnosti a bude obsahovat jen nejdůležitější a nejzákladnější informace a vyobrazení, protože detailnější informace by dalece přesahovaly zadání a očekávaný rozsah této práce.

Struktura projektu je patrná na Obrázku 12. Projekt umožňuje načtení i více rozhraní, ve kterých se po otevření nachází seznam jednotlivých dostupných operací.

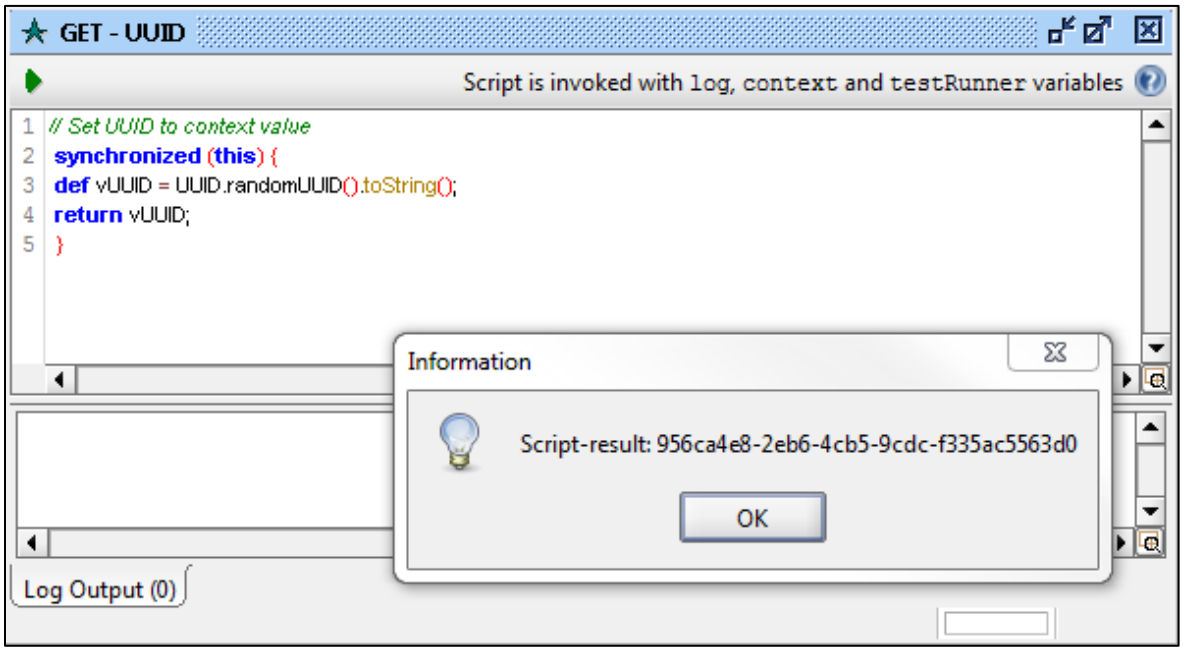
V rámci projektu se dále vytváří jednotlivé testovací balíky, které mohou obsahovat i několik testovacích případů. Testovací balíky tedy umožňují logicky rozdělit jednotlivé testované případy nadefinované v rámci testovacího scénáře a rozčlenit projekt do přehlednější struktury.

Každý testovací případ se skládá z testovacích kroků. Testovací kroky jsou uvedeny v určitém pořadí a realizují tak potřebný testovací scénář. Testovací kroky bývají nejčastěji realizovány ve formě volání jednotlivých operací rozhraní, ale lze přidat i pomocné kroky jako například skripty v jazyce Groovy. Ty lze použít např. při generování unikátních identifikátorů, tzv. GUID (viz Obrázek 13) nebo získání systémového času. Mezi další kroky patří připojení ke zdrojům dat (data source), časové zpoždění (delay) a další.



Obrázek 12 - Struktura projektu

1. Projekt, 2. Rozhraní (Interface), 3. Testovací balík (Test suit), 4. Testovací případ (Test case), 5. Skript v jazyce Groovy, 6. Testovací krok (Test step), 7. Zátěžové testy (Load tests).

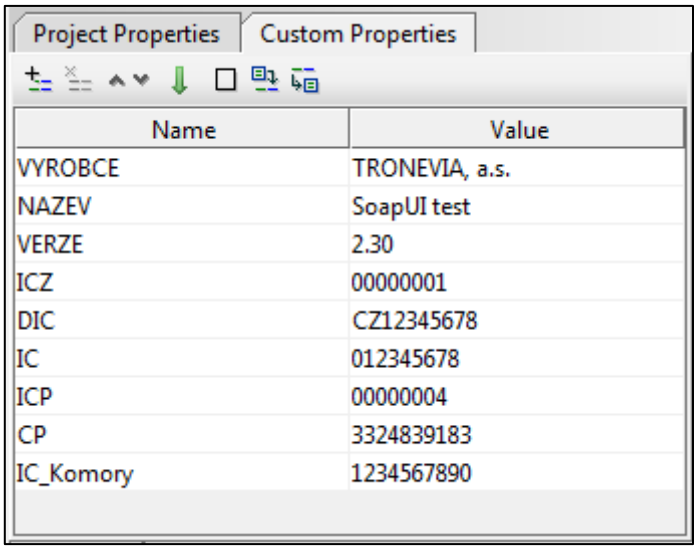


Obrázek 13 – Pomocný testovací krok obsahující skript pro generování unikátního identifikátoru

Při volání jednotlivých operací je možné používat proměnné, které mohou nabývat předdefinovaných konstantních hodnot nebo hodnot dynamických. Konstantní hodnoty (např. přihlašovací údaje) jsou typicky známé již při vytváření a spouštění testů. Hodnoty dynamických proměnných při vytváření nebo spouštění testů známé nejsou, typicky se jedná např. o návratové hodnoty z odpovědi na volání funkce.

Proměnné mohou mít různý rozsah platnosti:

- Globální: viditelné pro všechny právě otevřené projekty.
- Na úrovni projektu: viditelné pouze pro projekt, ve kterém jsou definovány, viz Obrázek 14.
- Na úrovni balíků: viditelné pouze pro testovací balík (test suit), ve kterém byly definovány.
- Na úrovni případů: viditelné pouze pro testovací případ (test case), ve kterém byly definovány.



Obrázek 14 - Seznam proměnných vytvořených v rámci jednoho projektu

Struktura okna testovacího kroku obsahujícího operaci je patrná na Obrázku 15. Okno obsahuje přístup k mnoha dalším funkcionalitám, vybrány byly však pouze ty nejdůležitější.

Koncovým bodem (endpoint) je myšleno cílové umístění služby, kam je dotaz zasílán. Lze ho nastavit pro každou testovanou operaci individuálně, ale vhodnější je nastavení koncového bodu přímo ve vlastnostech příslušného rozhraní, odkud je pak automaticky načten, pokud není zvoleno jinak. V tomto případě je navíc první část adresy načítána z globální proměnné.

Okno dále obsahuje vyobrazení dotazu operace. Jsou zde nabídnuty 4 možnosti zobrazení:

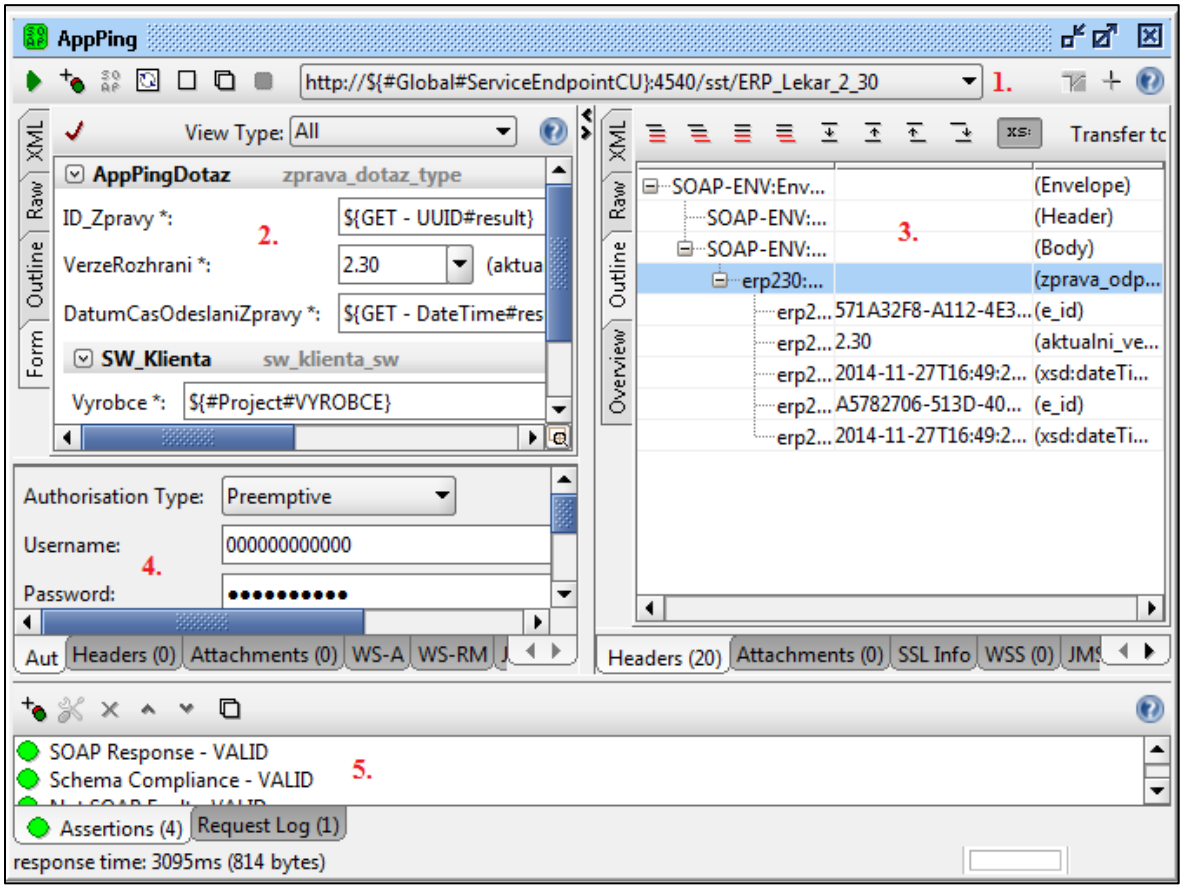
- XML, kde je dotaz vyobrazen ve struktuře XML dokumentu a umožnuje i nahlédnutí do odpovídajícího XSD.
- Raw, který zahrnuje kompletní dotaz a je vhodný zejména v situacích, kdy je zapotřebí zobrazit HTTP hlavičky odeslaného dotazu.
- Outline, který přehledně vyobrazuje strukturu jednotlivých elementů.
- Form, který je díky formulářovému vyobrazení vhodný pro vyplnění dotazu.

Po odeslání dotazu se v případě validního vyplnění a splnění bezpečnostních podmínek vrátí odpověď. Odpověď není možné editovat, je možné pouze její prohlížení. Opět je zde nabídnuto více možností zobrazení odpovědi. První tři jsou obdobné, čtvrtá možnost (Overview) je graficky podobná formulářovému vyobrazení z dotazu, avšak neumožnuje editaci.

Pod sekcí dotazu je k nalezení záložka pro vyplnění autorizace.

Poslední důležitou sekcí je sekce Assertions, která nám umožnuje přidat tvrzení o odpovědi, která musí být splněna, aby výsledek testu byl vyhodnocen jako pozitivní. Mezi nejpoužívanější patří například:

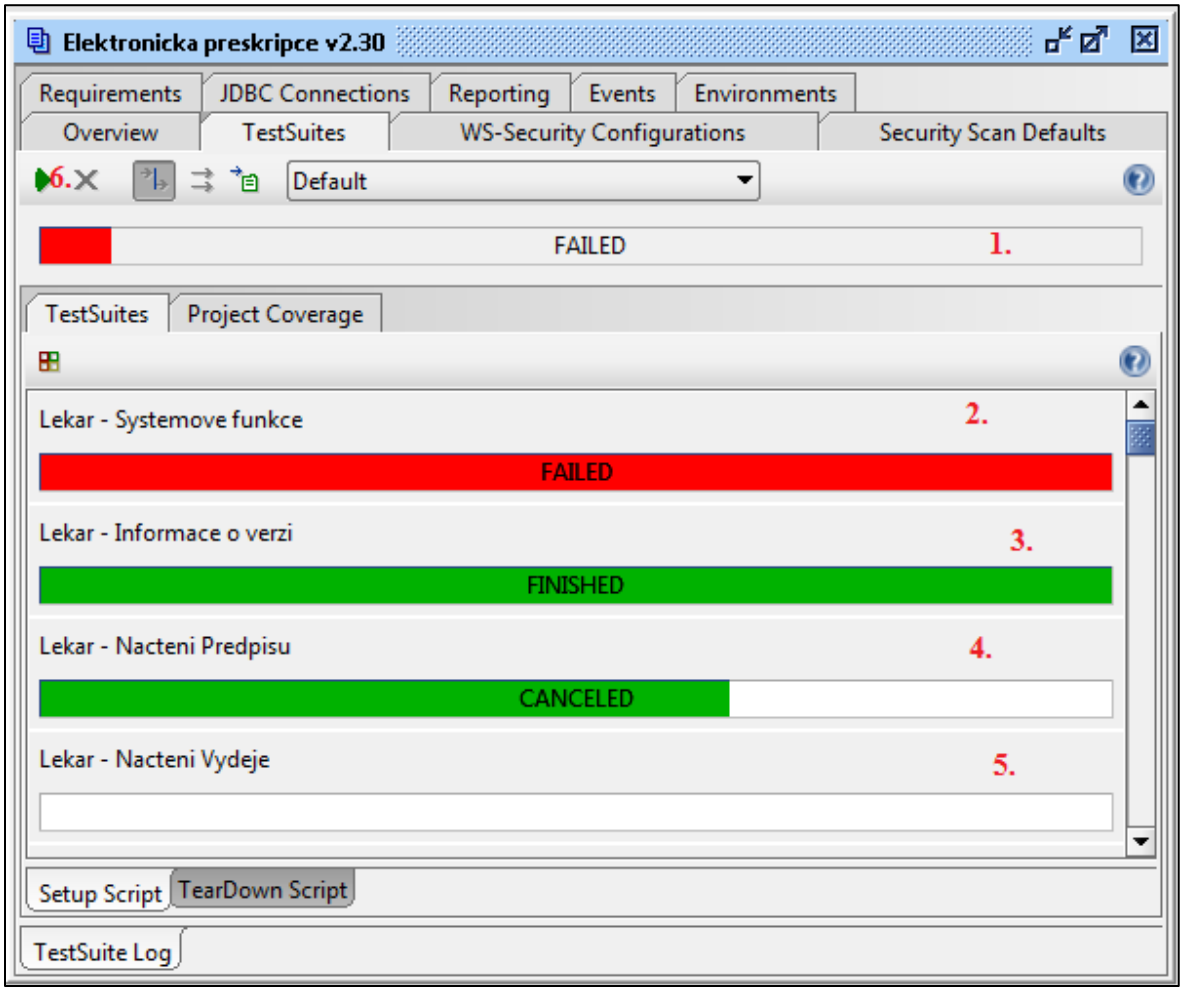
- Not Soap Fault, které tvrdí, že nesmí být vrácena chyba SOAP.
- Schema Compliance, které hlídá, aby příchozí odpověď dodržela omezení definovaná v XSD.
- XPath Match, kontroluje, že výsledek definovaného XPath výrazu odpovídá očekávané hodnotě.



Obrázek 15 - Struktura okna pro testovací krok operace

1. Koncový bod (Endpoint), 2. Dotaz (Request), 3. Odpověď (Response), 4. Autentizace/Autorizace, 5. Tvrzení (Assertions)

Testy je možné spouštět po jednotlivých krocích, případech, balících nebo jako celý projekt (viz Obrázek 16), přičemž vyobrazení je vždy podobné.



Obrázek 16 - Průběh všech testů v projektu

1. Výsledný stav testování. 2. Test skončil negativním výsledkem. 3. Test skončil pozitivním výsledkem. 4. Test byl přerušen. 5. Test zatím neproběhl. 6. Spustit testy.

5.3 Vyhodnocení průběhu testování

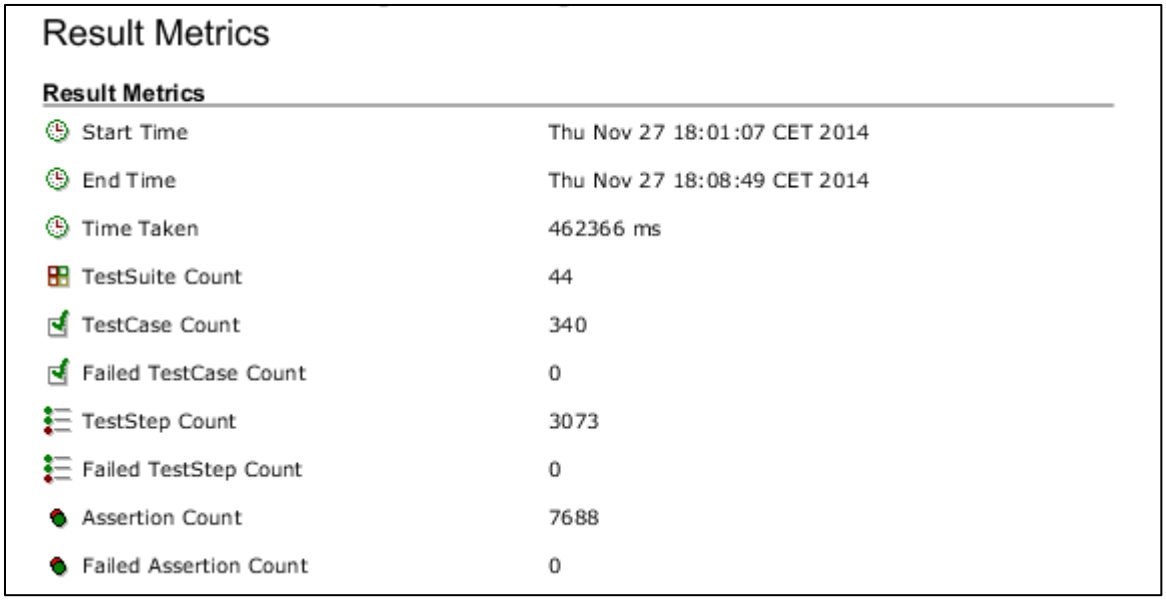
V případě, že jsou při běhu testů vráceny chyby, je potřeba tyto chyby projít jednu po druhé a vyhodnotit nejpravděpodobnější původ jejich vzniku. Zpočátku vzniká většina chyb na straně programátora. Chyby jsou zaevidovány (v našem případě v helpdesku) a jejich seznam předložen programátorovi s žádostí o jejich odstranění. Po odstranění chyb programátor informuje testera (a připíše případné komentáře k jednotlivým incidentům na helpdesk). Následně jsou znovu spuštěny a ověřeny zmíněné problémové sekce. Dalším častým původem chyb je zadávání nevalidních dat při vyplňování testovaných vstupů. Takovéto chyby jsou většinou jednoduše a rychle rozpoznatelné a odstranitelné. Jejich odstranění je záležitostí testera. Nejméně časté jsou pak chyby na straně analytika. Oprava chyb se opakuje až do doby, než veškeré testovací scénáře proběhnou s pozitivními výsledky. V tu chvíli je možné vygenerovat report o průběhu testování a přejít k zátěžovému testování. Poté je projekt uložen a v pravidelných intervalech spouštěn (automatické testování), což zaručuje pravidelnou kontrolu chodu WS.

5.4 Generování reportu o průběhu testování

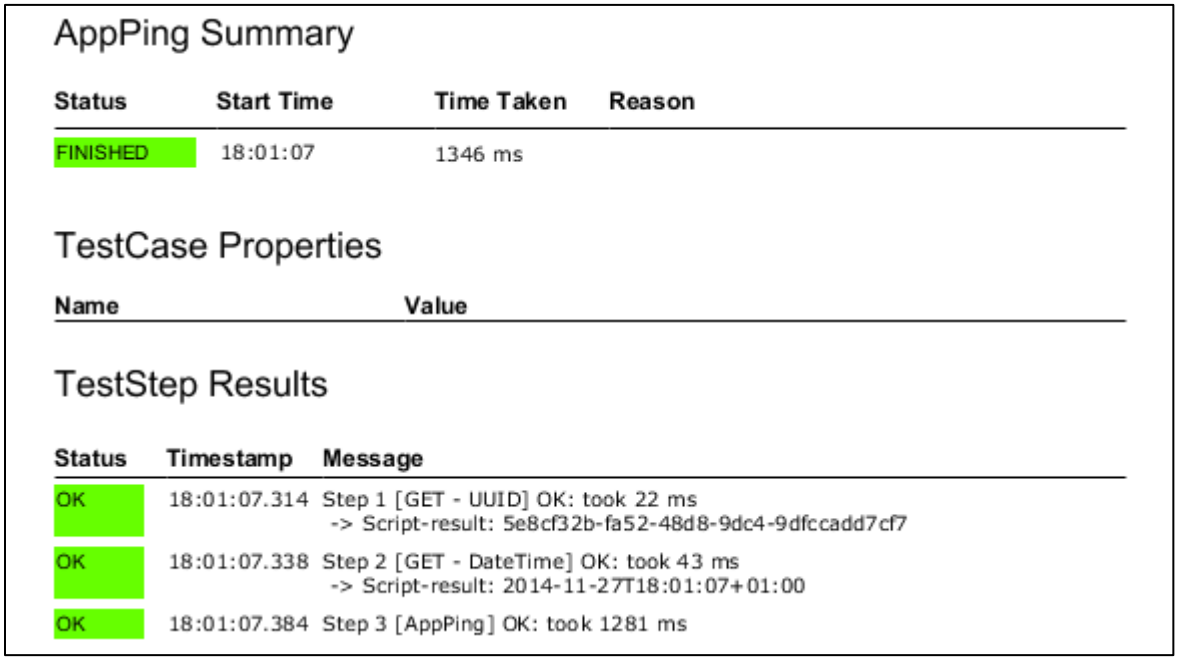
Velmi důležitou funkcionalitou tohoto testovacího nástroje je možnost vytvoření reportu o průběhu a výsledku provedených testů, který detailně vyobrazuje informace o projektu a:

- proměnných,
- rozhraních a koncových bodech,
- balíkách, případech, krocích,
- časech zpracování odpovědí,
- výsledných statusech odpovědí.

Vybrané části reportu jsou k vidění na Obrázku 17 a Obrázku 18.



Obrázek 17 - Ukázka z reportu o průběhu testování: souhrnné výsledky testů



AppPing Summary

Status	Start Time	Time Taken	Reason
FINISHED	18:01:07	1346 ms	

TestCase Properties

Name	Value

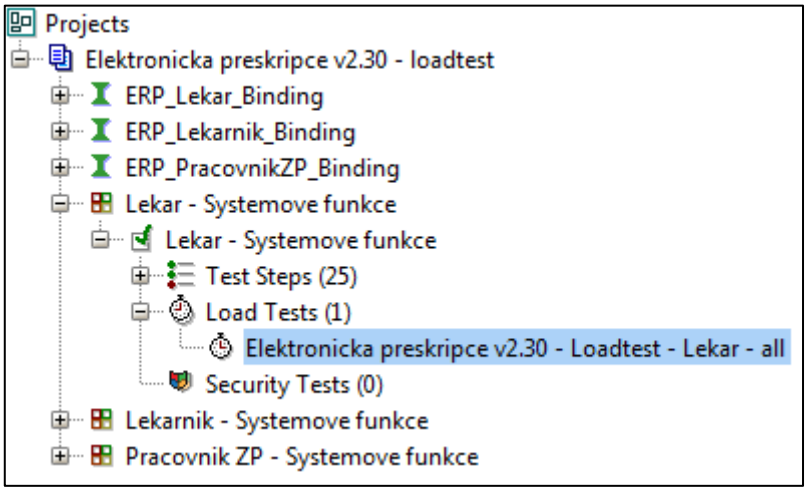
TestStep Results

Status	Timestamp	Message
OK	18:01:07.314	Step 1 [GET - UUID] OK: took 22 ms -> Script-result: 5e8cf32b-fa52-48d8-9dc4-9dfccadd7cf7
OK	18:01:07.338	Step 2 [GET - DateTime] OK: took 43 ms -> Script-result: 2014-11-27T18:01:07+01:00
OK	18:01:07.384	Step 3 [AppPing] OK: took 1281 ms

Obrázek 18 - Ukázka z reportu průběhu testování: výsledky jednotlivých kroků v rámci jednoho testovacího případu

5.5 Zátěžové testování

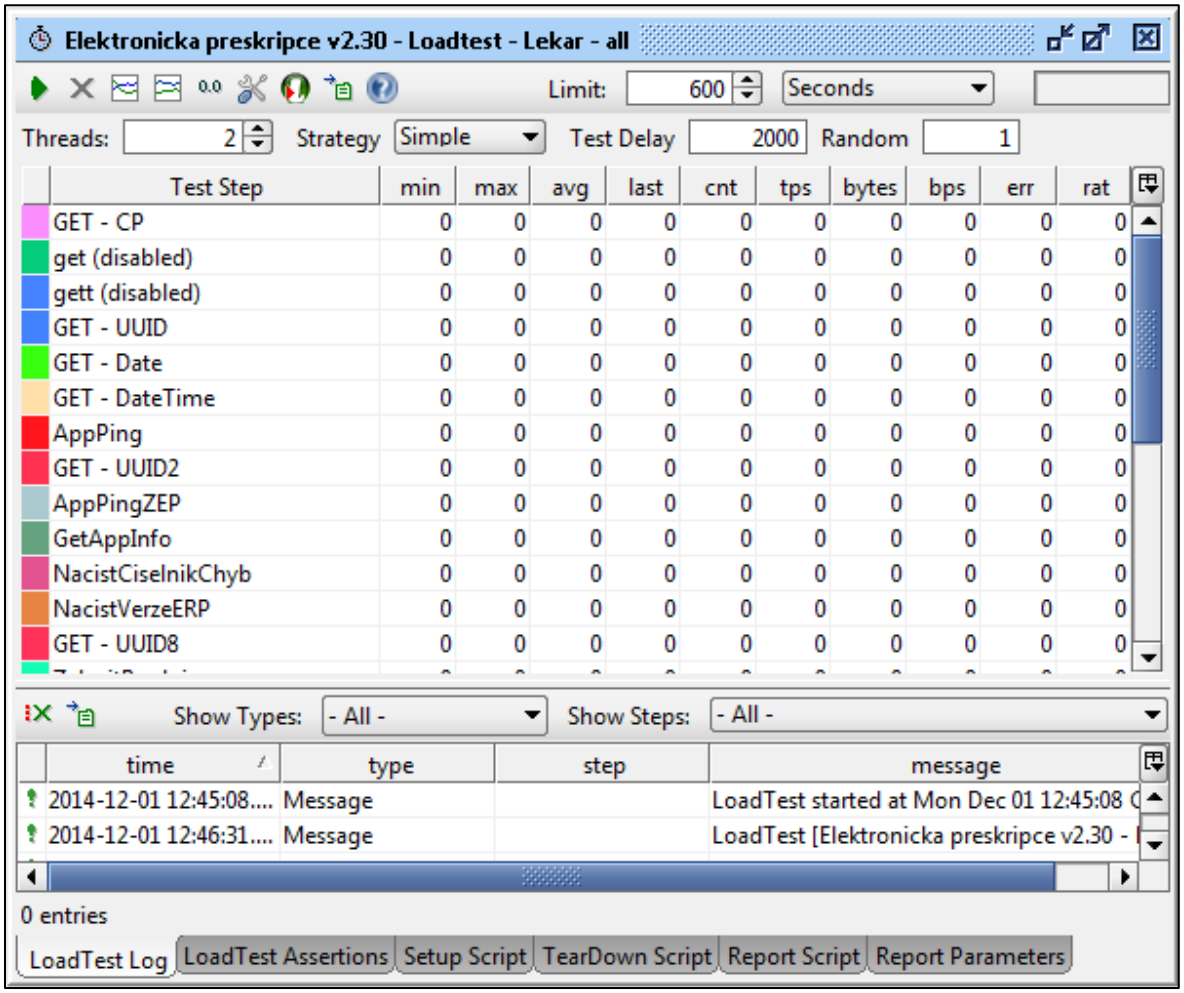
Zátěžové testy můžeme vložit přímo do scénářů obsahujících jednotlivé kroky funkčního testování. Nicméně vhodnější postup je takový, kdy dojde k vytvoření samostatného projektu nebo alespoň balíčku/případu, obsahujícího sled kroků, nejlépe vystihujících reálné použití aplikace. Po vytvoření takového scénáře v testovacím případu bude tento případ doplněn o zátěžový test (viz Obrázek 19).



Obrázek 19 - Struktura testovacího případu určeného pro zátěžové testování

Po otevření zátěžového testu se zobrazí okno, které nám umožní nastavení času, počtu vláken, zpoždění mezi jednotlivými cykly a další (viz Obrázek 20). Některé hodnoty je možné po čas testování upravovat (např. počet vláken, zpoždění mezi cykly aj.).

Při běhu jsou navíc k dispozici grafy a číselná znázornění průběhu. Po doběhnutí testů je opět možné vygenerovat report, který obsahuje obdobné informace, jako jsou k dispozici v pracovním oknu.



Obrázek 20 - Pracovní okno pro zátěžové testování

5.6 Řešení univerzálnosti testů pro více prostředí

Jak bylo zmíněno výše, je potřeba testovat prostředí testovací i produkční. Nabízejí se dvě řešení. První řešení je takové, že dojde k vytvoření dvou totožných projektů s rozdílnou sadou URL a přihlašovacích údajů. Toto řešení ale není vhodné, protože při každé úpravě testů je potřeba zohlednit tuto úpravu v obou prostředích. Proto se nabízí druhé, vhodnější řešení, kdy pomocí proměnných a skriptů umožníme v rámci jednoho projektu „přepínat“ mezi testovacím a produkčním prostředím.

Při startu programu je ze souboru typu properties načtena proměnná, která rozliší prostředí, na které se mají zasílat dotazy WS. Následně je při otevírání projektu načtena sada proměnných, obsahujících přihlašovací údaje, URL a jiné hodnoty, které jsou odlišné v závislosti na zvoleném prostředí. V druhé části skriptu jsou pak doplněny přihlašovací údaje do jednotlivých dotazů v závislosti na rozhraní, pod které spadají (viz Obrázek 21).

```
for(testSuite in project.getTestSuiteList()){
  for( testCase in testSuite.getTestCaseList()) {
    log.info("Setting HTTP basic auth for all WSDL test requests in test case ["+ testCase.getLabel()+"]")
    for(testStep in testCase.getTestStepList()) {
      if(testStep instanceof WsdlTestRequestStep) {
        def bindingName = testStep.getHttpRequest().getInterface().getBindingName().toString()
         bindingName = bindingName.substring(bindingName.indexOf("}")+1)
        testStep.getTestRequest()
       .setUsername(com.eviware.soapui.SoapUI.globalProperties.getPropertyValue(bindingName + "_ID" ))
         testStep.getTestRequest()
       .setPassword(com.eviware.soapui.SoapUI.globalProperties.getPropertyValue(bindingName + "_PWD" ))
       }
     }
   }
}
```

Obrázek 21 - Ukázka části skriptu pro vyplnění přihlašovacích údajů

Závěr

Tato práce se zabývá základní teorií automatizovaného testování v průběhu procesu tvorby softwaru. Shrnuje význam testování jako takového, i mechanismy a typy chyb, které se v průběhu vzniku softwarové aplikace mohou vyskytnout. Zároveň klasifikuje testy do skupin podle jejich povahy a účelu. Popisuje i zásady organizace samotné tvorby v rámci větších vývojových týmů.

V potřebném rozsahu práce přibližuje i technologii webových služeb, základní protokoly používané ve webových službách a principy jejich tvorby.

Stěžejní prostor je věnován srovnání čtyř různých nástrojů používaných pro testování webových služeb (SoapUI, SOAPSonar, Examine a Storm). Srovnání je realizováno z různých pohledů, mezi něž patří funkce, které tyto nástroje poskytují, jejich přehlednost, rychlost, s jakou je možné se funkce těchto nástrojů naučit, zabezpečení, doplňkové funkce atd.

Jako nejvhodnější nástroj ze srovnání vyšla volně dostupná verze softwaru SoapUI. Komerční verze tohoto nástroje byla následně využita v praktické části této práce, která spočívá v implementaci různých typů testů pro aplikaci ERP verze 2.30, která řeší problematiku elektronických receptů v rozsahu definovaném v zákonu o léčivech.

Kromě popisu tvorby těchto testů jsou v textu prezentovány i další aspekty, jako je zátěžové testování, generování protokolů z testování, aplikace vytvořených testů v různých prostředích apod.

Literatura

[1] BOROVCOVÁ, Anna. *Testování webových aplikací: Část II: Základy testování* [online]. 2011 [cit. 30. 3. 2014]. Dostupné z: http://testovanisoftwaru.blogspot.com/p/zakladytestovanipdf.html

[2] PATTON, Ron. *Testování softwaru*. Vyd. 1. Praha: Computer Press, 2002, xiv, 313 s. Programování. ISBN 80-722-6636-5.

[3] Druhy chyb. *SW Testování* [online]. [cit. 20. 11. 2014]. Dostupné z: http://www.swtestovani.cz/index.php/uvod-do-testovani/10-druhy-chyb

[4] HLAVA, Tomáš. Progresní a regresní testy. *Testování softwaru* [online]. 2011 [cit. 8. 4. 2014]. Dostupné z: http://testovanisoftwaru.cz/tag/regresni-testy/

[5] HLAVA, Tomáš. Funkční a nefunkční testy. *Testování softwaru* [online]. 2011 [cit. 8. 4. 2014]. Dostupné z: http://testovanisoftwaru.cz/tag/nefunkcni-testy/

[6] Testování softwaru. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2014-12-02]. Dostupné z: http://cs.wikipedia.org/wiki/Testov%C3%A1n%C3%AD_softwaru

[7] Testovací dokumentace - plán, scénář, případ. *SW Testování* [online]. [cit. 19. 11. 2014]. Dostupné z: http://www.swtestovani.cz/index.php?option=com_content&view=article&id=15:testovaci-dokumentace-plan-scena-pipad&catid=3:zaklady&Itemid=11

[8] CHAPPELL, David A a Tyler JEWELL. *Java Web services*. 1st ed. Sebastopol, CA: O'Reilly, c2002, xii, 262 p. ISBN 05-960-0269-6.

[9] MLÝNKOVÁ, Irena a Jaroslav POKORNÝ. *XML technologie: principy a aplikace v praxi*. 1. vyd. Praha: Grada, 2008, 267 s. Průvodce (Grada). ISBN 978-80-247-2725-7.

[10] MALÝ, Martin. REST: architektura pro webové API. *Zdroják: o tvorbě webových stránek a aplikací* [online]. 2009 [cit. 4. 6. 2014]. Dostupné z: http://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/

[11] KUBA, Martin. Web Services. In *DATAKON 2006, Proceedings of the Annual Database Conference*. Brno, 2006. s. 93-112, 20 s. ISBN 80-210-4102-1.

[12] Envelope. *XML Design Patterns* [online]. [cit. 16. 2. 2014]. Dostupné z: http://www.xmlpatterns.com/EnvelopeMain.shtml

[13] SOAP UI. *SW Testování* [online]. [cit. 8. 4. 2014]. Dostupné z: http://www.swtestovani.cz/index.php?option=com_content&view=article&id=44:soap-ui&catid=14:testovaci-nastroje&Itemid=31

[14] Compare SoapUI and SoapUI Pro. SMARTBEAR SOFTWARE. *SoapUI: The Home of Functional Testing* [online]. [cit. 15. 10. 2014]. Dostupné z: http://www.soapui.org/Go-Pro/compare-soapui-and-soapui-pro.html

[15] SOAPSonar Personal Edition (Free). CROSSCHECK NETWORKS. *Service and API Testing, Service and API Virtualization, Service and API Gateway Security* [online]. [cit. 11. 4. 2014]. Dostupné z: http://www.crosschecknet.com/products/soapsonardetails_personal.php