VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

TESTOVÁNÍ WEBOVÝCH SLUŽEB

SEMESTRÁLNÍ PROJEKT
TERM PROJECT

AUTOR PRÁCE
AUTHOR

Ing. RADIM REŠ

BRNO 2015

Obsah

1 Úvod **2**

2 Webové služby **3**
2.1 Dokument XML jako datový nosič . . . 3
2.1.1 Co je XML? . . . 3
2.1.2 Formátování dokumentu (XSL a XSLT) . . . 5
2.2 Webové služby a jejich protokoly . . . 6
2.2.1 Co je webová služba? . . . 6
2.2.2 UDDI . . . 7
2.2.3 SOA . . . 7
2.2.4 Protokol SOAP . . . 7
2.2.5 WSDL a jeho styly . . . 8
2.2.6 Protokol HTTP (REST) . . . 8
2.2.7 Mikroslužby . . . 9

3 Testování software **11**
3.1 Pohled na systém . . . 11
3.1.1 Bílá skříňka . . . 11
3.1.2 Šedá skříňka . . . 11
3.1.3 Černá skříňka . . . 12
3.2 Úrovně testování . . . 12
3.2.1 Jednotkové testování . . . 13
3.2.2 Integrační testování . . . 13
3.2.3 Systémové testování . . . 15
3.2.4 Akceptační testování . . . 15
3.3 Testování webových služeb . . . 15
3.3.1 API . . . 16
3.3.2 Testování s využitím analýzy hraničních hodnot . . . 16

4 Nástroje automatického testování **17**
4.1 Soap UI . . . 17
4.2 Selenium . . . 18
4.3 Sikuli . . . 18

5 Závěr **20**

Kapitola 1

Úvod

S příchodem komplexních a stále nových a nových informačních systémů podobného typu nastává požadavek na modularitu a škálovatelnost. Důvodem je především znovupoužitelnost již hotových komponent. Tuto znovupoužitelnost nám zajišťuje bezstavovost a tím i nezávislost komponent, které spolu komunikují pomocí webových služeb. Tato práce se tedy zabývá architekturou orientovanou na služby a specializuje se na testování webových služeb.

Webové služby využívají XML formát k přenosu dat, tudíž začneme právě tímto datovým nosičem a popíšeme si, co je to XML a jak vypadá jeho struktura. Zajímat nás bude, kde lze XML aplikovat a na okraj si ukážeme i jak lze XML uspořádat a formátovat pomocí XSL a XSLT. Tím porozumíme XML jako datovému nosiči webových služeb. U webových služeb nás dále budou zajímat přístupy a protokoly pro jejich implementaci. Tím se dostaneme k pojmům, jako jsou SOAP, WSDL, REST a na okraj si uvedeme něco k novince, kterou jsou mikroslužby.

V případě testování software si ukážeme pohledy na systém pro vymezení a lepší pochopení volby vhodné úrovně testování pro případ pokrytí webových služeb testem. Dále využijeme znalostí o webových službách a základních znalostech o testování a přistoupíme k samotnému testování webových služeb s využitím analýzy hraničních hodnot na ekvivalenčních třídách parametrů, kde se dozvíme, jak určit validní a nevalidní testové případy.

Na závěr si představíme nástroje automatického testování, které by mohly být vhodné pro pokrytí webových služeb automatickým testem. Nejdříve se podíváme na nejnižší možnou úroveň testování webových služeb, kterou jsou integrační testy pokryté nástrojem Soap UI. Další dva nástroje jsou již o úroveň výše. Oba se pohybují na úrovni systémových a akceptačních testů. Jedním z nich je známý nástroj Selenium zaměřující se na testování webových stránek prostřednictvím webového prohlížeče. Následně se dostaneme i k nově vyvíjenému nástroji Sikuli, jenž se specializuje na manipulaci s uživatelským rozhraním i mimo webové prohlížeče pomocí rozpoznávání obrazových vzorů.

Kapitola 2

Webové služby

2.1 Dokument XML jako datový nosič

Tato sekce čerpá z knihy [1]. Zde si popíšeme, co je to XML a jaká je jeho struktura, kterou si rozlišíme na logickou a fyzickou. Zajímat nás také bude, kde lze XML aplikovat a na okraj si ukážeme i jak lze XML uspořádat a formátovat pomocí XSL a XSLT. Ve výsledku porozumíme XML jakožto datovému nosiči využívaného webovými službami, kterými budeme pokračovat.

2.1.1 Co je XML?

Název XML vznikl jako zkratka z anglického *Extensible Markup Language*. Tento jazyk byl vyvinut konsorciem W3C a vytvořen podle zkušenosti s předchozími značkovacími jazyky. XML je ideální formát pro ukládání strukturovaného a semi-strukturovaného textu určeného pro šíření a konečnou publikaci na celé řadě médií. Dokument XML obsahuje specifické instrukce, nazývané tagy (značky), elementy a entity, které ohraničují části dokumentu. Původně byl jazyk XML navržen pro oblast publikování, ale je také vhodný i pro výměnu informací mezi programy a počítačovými systémy, kde je možné jednoznačné identifikování komplexních datových struktur:

```
<RESPONSE>
    <MESSAGE>
        <ID>2</ID>
        <CORRELATIONID>1</CORRELATIONID>
    </MESSAGE>
    <PRODUCT id="1">
        <CATEGORY>Nábytek z masivu</CATEGORY>
        <SUBCATEGORY>Postele</SUBCATEGORY>
        <NAME>Postel Mark</NAME>
        <PRICE>3000</PRICE>
        <DESCRIPTION>Postele s roštem, bez doplňků</DESCRIPTION>
        <PARAMETERS>
            <PARAM id="width">1000</PARAM>
            <PARAM id="length">2050</PARAM>
            <PARAM id="height">355</PARAM>
        </PARAMETERS>
```

```
    </PRODUCT>
</RESPONSE>
```

Dokument XML má logickou a fyzickou strukturu. Logická struktura rozděluje dokument do pojmenovaných jednotek a podjednotek, nazývaných elementy. Fyzická struktura umožňuje pojmenovat a uložit samostatné části dokumentu, zvané entity, někdy i v různých datových souborech, aby mohly být tyto informace opakovaně použity. Logická struktura zahrnuje celou řadu omezení, která musejí být dodržena ve všech validních dokumentech XML. Správnost dokumentu můžeme otestovat pomocí XML parseru, jenž je volně dostupný na internetu a vlastní ho i speciální XML editory. XML je ve skutečnosti metajazyk, což znamená, že je to jazyk, který se používá k popisu dalších jazyků. Neexistuje předdefinovaný seznam elementů. XML poskytuje naprostou svobodu při využívání prvků, jejichž jména mají pro danou aplikaci smysl. Existuje však i mechanismus, který nám umožňuje předdefinování elementů, jenž mohou být použity v dané třídě dokumentů. DTD (*Document Type Definition*) definuje povolené prvky. Validující parser porovnává pravidla DTD s příslušným dokumentem, aby určil, zda dokument těmto pravidlům neodporuje. Tato vlastnost dovoluje vytvářet softwarové filtry pro překlad mezi různými typy dokumentů, přičemž je zajištěno, že značkování použité pro přípravu dokumentu odpovídá známé specifikaci a je konzistentní. Řada průmyslových standardů pro výměnu a publikaci dat může být definována pomocí vhodného DTD. Následující příklad definuje značky *warning*, *section*, *image*, *emph* a *keyword* pro použití na příslušných místech ve všech relevantních dokumentech:

```
<!ELEMENT warning  (section*, image?)>
<!ELEMENT section  (#PCDATA | emph | keyword)*>
<!ELEMENT image  EMPTY>
<!ELEMENT emph   (#PCDATA)*>
<!ELEMENT keyword  (#PCDATA)*>
```

XML podporuje použití elementů se jmény, která popisují podstatu objektu, místo aby popisovaly, jak mají být tyto elementy zobrazeny či vytištěny. Oproti tradičním instrukcím založených na určení stylu má toto zobecněné značkování (*generalized markup*) jednu podstatnou výhodu. Taková informace je samopopisující, tedy může být vyhledána, vyjmuta a zpracována dle potřeby.

XML je možné aplikovat všude tam, kde je potřeba výměna komplexních dat mezi dvěmi systémy. XML se již nyní běžně aplikuje v mnoha oblastech, jakou jsou push technologie, EDI (*Electronic Data Interchange*) a obecné aplikace pro metadata (MCF, XML-Data a RDF). XML může být také použito jako výměnný formát pro relační databázové systémy. V tomto případě jsou tagy XML použity během přenosu záznamů, polí a vztahů mezi systémy jako vhodný obal dat. Strukturovaný a zároveň opakovatelný přístup XML je ideální pro ukladání řady souvisejících datových polí především v případě dat vybíraných z více tabulek spojených vztahem 1:N. Jak už tomu bývá zvykem, tak si jednotlivé společnosti vyvíjí vlastní standardy v tomto případě pro výměnu metadat. Příkladem jsou jmenovitě Microsoft (XML-Data), Netscape (MCF) a v poslední době i W3C (RDF). Jedno mají však všechny společné, cílem je definovat obecné přístupy k vytvoření samopopisných dat tím, že přiřadí další významnou úroveň značkování XML.

Technologie EDI je založena na lokálních řešeních. Jejím cílem je nejen zlevnit řešení pro elektronický obchod, ale také standardizace, která je v této oblasti velmi potřebná a žádaná. Existují dva přístupy komunikace server - klient:

1. **pull** Tradiční web je popisován jako technologie „tažení“ (pull), protože webový prohlížeč „stahuje“ HTML dokumenty z webového serveru, který pouze naslouchá a čeká na požadavky na webové stránky či jiný obsah. Webový klient je tedy aktivní a server je pasivní.

2. **push** Konceptce „tlačení“ (push) je založena na nápadu, že webový server bude posílat (tlačit) data každému klientskému programu, aniž by čekal na jejich požadavek. Klientský prohlížeč je dynamicky obnovován, jak se mění zasílané informace. Takovým příkladem je technilogie CDF společnosti Microsoft.

Díky zvyšujícímu se zájmu o technologii skriptování na straně klienta, se server může více soustředit na svůj základní účel, kterým je poskytování softwaru a dat připojenému klientovi. Například aplikace na straně klienta může číst data v XML, dávat jim význam a smysluplně reagovat na operace požadované apletem. Může některá data skrývat a opět zobrazovat, vytvářet obsahy a indexy, nebo jinak uspořádat zaslané informace podle mnoha různých kriterií, a to vše bez další interakce s webovým serverem, který pouze zaslal data.

2.1.2 Formátování dokumentu (XSL a XSLT)

Již víme, co je to XML a nyní budeme chtít tento dokument uspořádat a formátovat pro prezentaci. Dokumenty XML jsou určeny pro snadné čtení jak ze strany uživatele, tak aplikací. Neočekává se však, že uživatelé, kteří se zajímají o obsah těchto dokumentů, chtějí vidět tagy. Pro prezentaci informací ve formátu XML je proto nezbytné nahradit tagy příslušnými styly textu. K tomu slouží XSL (*eXtensible Stylesheet Language*) jako standard stylu a XSLT (eXtensible Stylesheet Language Transformations) jako standard pro transformaci struktur XML. Oba standardy dohromady tvoří doplněk k XML. Obsah elementu nemá explicitně žádný styl a textový editor jej tudíž zobrazí standardním písmem. Jeden styl písma běžně není akceptovatelný pro celou publikaci, kde uživatelé očekávají různorodost a vhodná vizuální vodítka naznačující strukturu informací.

Pro ilustraci mějme následující fragment XML dokumentu:

```
<headline>Příklad stylu</headline>
<introduction><section>Tento příklad ukazuje, jak důležité jsou styly pro
čitelnost materiálu.</section></introduction>
<section>Toto je <bold>běžný</bold> odstavec.</section>
<warning><section>Styly jsou důležité!</section></warning>
```

Kromě odstranění XML tagů není následující ukázka nijak formátována:

```
Příklad stylu Tento příklad ukazuje, jak důležité jsou styly pro čitelnost
materiálu. Toto je běžný odstavec. Styly jsou důležité!
```

S využitím stylů můžeme tento fragment zformátovat například takto:

Příklad stylu

Tento příklad ukazuje, jak důležité
jsou styly pro čitelnost materiálu.
Toto je **běžný** odstavec.

VAROVÁNÍ: Styly jsou důležité!

Jedna definice stylů může být sdílena řadou dokumentů, čímž se snadněji dosahuje stejného vzhledu u souvisejících dokumentů. Protože pravidla stylů jsou aplikována na specifické elementy, jsou styly úzce svázány s DTD. Dokument XML může být asociován s více než jedním souborem stylů. Nabízí se tedy kategorizace a znovupoužitelnost.

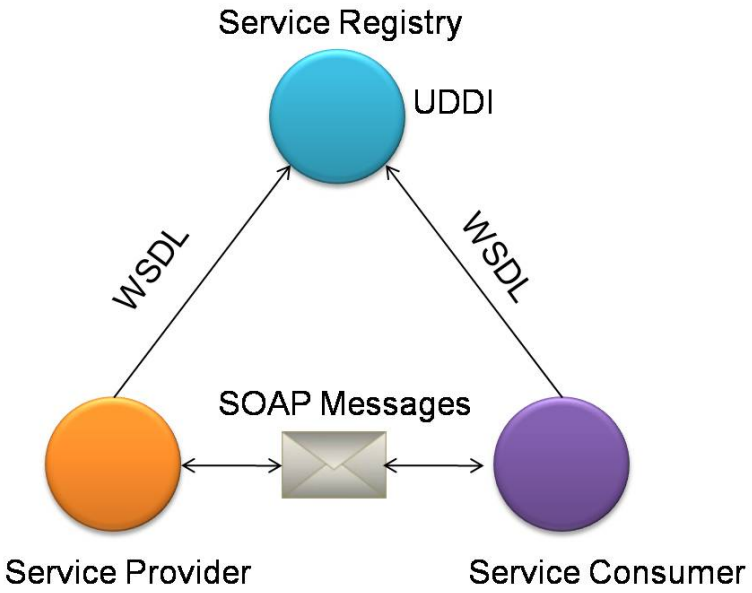
Standard XSL je v podstatě pouze sada formátovacích objektů, jako jsou bloky, inline objekty a buňky tabulky. Jelikož se jedná o abstraktní standard, mohou formátovací instrukce XSL nabývat různých forem, nejběžnější z nich je ovšem značkování XML, ve kterém elementy reprezentují typy objektů.

2.2 Webové služby a jejich protokoly

Tato kapitola čerpá z knih [11] a [12]. V této části nás bude zajímat, co je to webová služba. Jaké existují přístupy a protokoly pro implementaci webových služeb. Tudíž si uvedeme základní popis pojmů jako jsou SOAP, WSDL a REST a v závěru také základní informace k novince, kterou jsou mikroslužby.

2.2.1 Co je webová služba?

Webová služba je mechanismus poskytující standard pro klient - server komunikaci mezi variací aplikací pracujících na podobných nebo heterogenních platformách. Takto komunikující komponenty těchto aplikací si vyměňují data ve strukturovaných formátech. Samotný business proces často zahrnuje několikanásobné webové služby. Navíc tyto několikanásobé webové služby mohou být umístěny každá na jiném serveru patřícím rozdílným společnostem. Webové služby komunikují pomocí zpráv a dat ve sktrukturované formě, což konkrétně může být s využitím XML. Cílem takové komunikace je umožnit popis, zpřístupnění a volání webových služeb přes internet. Jak mechanismus webových služeb vypadá znázorňuje obrázek 2.1 níže.



Obrázek 2.1: Webové služby

2.2.2 UDDI

Název UDDI je zkratkou z anglického *Universal Description, Discovery and Integration*, což je nezávislá platforma registrů založených na XML, která zviditelňuje a lokalizuje webové služby na internetu. Také definuje, jak webové služby přes internet interagují. UUID byl původně navržen jako standard jádra webové služby, které SOAP zprávám zajišťuje přístup k WSDL dokumentům.

2.2.3 SOA

Zde jsou čerpány informace převážně z článku [7]. Název SOA je zkratkou z anglického *Service-Oriented Architecture*, nebo-li architektura orientovaná na služby. Tato architektura zahrnuje vícero systémů, jejichž komponenty poskytují služby. Tyto služby spolu komunikují prostřednictvím sítě. Tedy architektura SOA umožňuje rozčlenit složité systémy na podsystémy, které mohou být fyzicky umístěny kdekoliv v síti. Spolu související systémy následně můžeme na sebe skládat nebo je řadit do skupin. Pomocí architektury orientované na služby můžeme snadno zapouzdřit implementační detaily komponenty a pracovat s takovou komponentou na abstraktní úrovni. Díky tomu lze zajistit znovupoužitelnost komponent, které mohou být dostupné i pro jiné systémy v rámci jiných aplikací. Vnitřní logika každé komponenty by tedy měla být nezávislou pro správu zdrojů, ze kterých čerpá. Aby komponenta byla znovupoužitelná, musí být také bezstavová.

2.2.4 Protokol SOAP

Zde jsou čerpány informace převážně z článku [14]. Název SOAP je zkratkou z anglického *Simple Object Access Protocol*. SOAP pomáhá software předat a příjmout XML zprávy přes internet. Je tedy protokolem pro výměnu XML zpráv s využitím HTTP a díky tomu může snadno procházet přes firewall. Formát SOAP tvoří základní vrstvu komunikce mezi webovými službami a poskytuje prostředí pro tvorbu složitější komunikace. Existuje několik různých druhů šablon pro komunikaci na protokolu SOAP. Nejznámější z nich je RPC (*Remote Procedure Call*) šablona, která umožňuje aplikaci vykonávat vzdálenou proceduru v síti. Když obalíme již dříve zmíněný XML přiklad pomoci SOAP, dostaneme:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <RESPONSE>
        <MESSAGE>
            <ID>2</ID>
            <CORRELATIONID>1</CORRELATIONID>
        </MESSAGE>
        <PRODUCT id="1">
            <CATEGORY>Nábytek z masivu</CATEGORY>
            <SUBCATEGORY>Postele</SUBCATEGORY>
            <NAME>Postel Mark</NAME>
            <PRICE>3000</PRICE>
            <DESCRIPTION>Postele s roštem, bez doplňků</DESCRIPTION>
            <PARAMETERS>
                <PARAM id="width">1000</PARAM>
                <PARAM id="length">2050</PARAM>
```

```
            <PARAM id="height">355</PARAM>
          </PARAMETERS>
        </PRODUCT>
    </RESPONSE>
  </soap:Body>
</soap:Envelope>
```

2.2.5 WSDL a jeho styly

Zde jsou čerpány informace převážně z článku [3]. Název WSDL vznikl jako zkratka z anglického *Web Services Description Language*. WSDL je využito k popisu, jak přistupovat k webovým službám a jaké operace mohou provádět. Tento popis bývá zpravidla v XML formátu a popisuje SOAP komunikaci. Obecně můžeme říci, že WSDL popisuje, jak služba vázána na protokol zpráv, zejména k protokolu SOAP zpráv. Vazba mezi WSDL a SOAP může být buď vazba stylu RPC (z anglického *Remote Procedure Call*) nebo vazby dokumentového stylu. SOAP vazba může mít také kódované (*encoded*) použití nebo doslovné (*literal*) použití, což nám nabízí čtyři modely:

- RPC / encoded
- RPC / literal
- Document / encoded
- Document / literal

Když přidáme do této sbírky vzor, který se běžně nazývá *Dokument / literal wrapped pattern*, dostaneme poslední pátý model na výběr pro vytvoření nebo-li přesněji přeložení WSDL souboru.

2.2.6 Protokol HTTP (REST)

Zde jsou čerpány informace převážně z článku [13]. Název REST vznikl jako zkratka z anglického *Representational State Transfer*, jenž je zjednodušenou alternativou SOAP a WSDL. REST definuje množinu architektonických principů, kterými můžeme navrhnout webovou službu zaměřující se na systémové zdroje přímo nad HTTP protokolem. REST je v tomto případě bezstavový, ale jeho rozhraní zahrnuje stejné metody jako HTTP, jmenovitě:

- GET - Na základě požadavku popsaného v URL vrácí data.
- POST - Slouží především k odeslání dat, např. vyplněné ve formuláři nějakého uživatelského rozhraní. Data, která se nevejdou do parametrů v URL v případě GET, lze takto přenést metodou POST. Opět jde o požadavek ze strany klienta, který vrací data.
- PUT - Podobně jako POST slouží k odeslání dat. POST se používá pro vytváření dat, PUT se spíše používá pro jejich aktualizaci, ale není vyloučené i vytváření.
- DELETE - Maže záznam či skupinu záznamů.

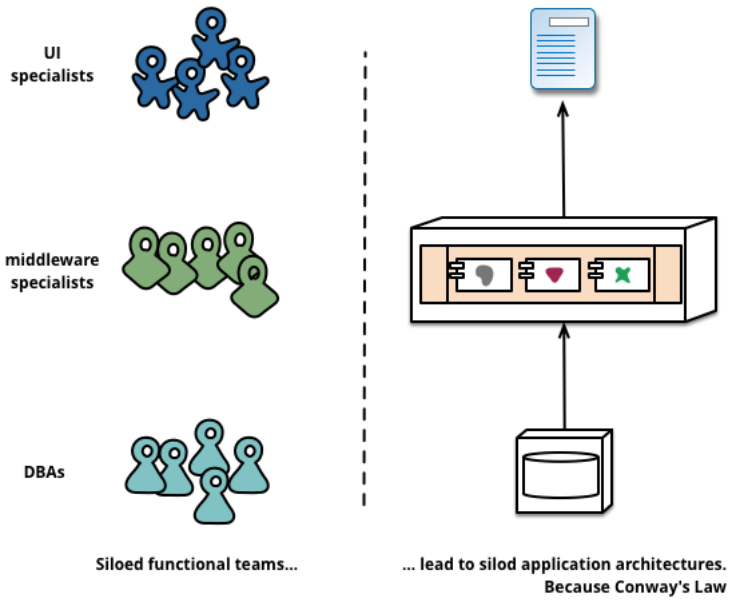
REST typicky vrací data ve formátu JSON, neboť je snadno následně zpracovatelný pomocí JavaScriptu v uživatelském rozhraní na straně klienta.

2.2.7 Mikroslužby

Zde jsou čerpány informace převážně z článku [6]. Mikroslužby jsou v oblasti softwarové architektury nový styl přistupu k vývoji samostatné aplikace jako k sadě malých služeb, jenž každá běží ve vlastním procesu a komunikuje často se svými zdroji přes API protokolu HTTP. Tyto služby jsou stavěny k naplnění obchodních potřeb a jsou také velmi snadno nasaditelné pomocí plné automatizace. Řízení těchto služeb je minimálně centralizované a takové služby mohou být zapsány v různých programovacích jazycích a použity na různých technologiích pro ukládání dat. Webové služby se často používají z důvodu rozdělení aplikace na jednotlivé znovupožitelné komponenty, z toho důvodu se zavádějí týmy specialistů na následujících 3 úrovních:

- Uživatelské rozhraní
- Databáze
- Rozhraní - zde se nachází experti, kteří se zabývají vývojem webových služeb

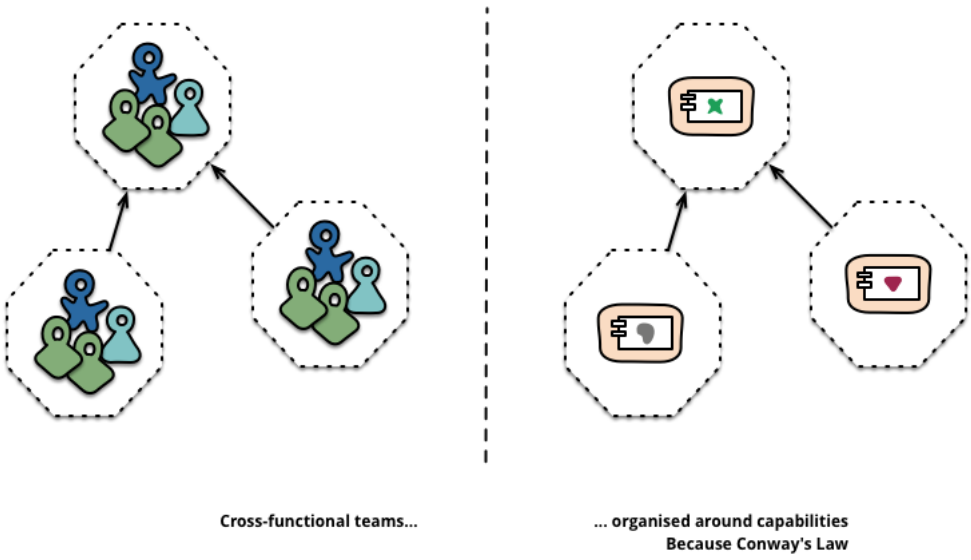
Více ilustruje obrázek 2.2 níže.



Obrázek 2.2: Webové služby - organizace týmu specialistů

Mikroslužby oproti běžným webovým službám umožňují organizovat týmy dle obchodních záměrů, což popisuje následující obrázek 2.3.

Týmy s různými funkcemi jsou odpovědné za provoz jednotlivých takto vzniklých částí produktů. Jednotlivé části produktů spolu komunikují prostřednictvím tzv. sběrnice zpráv. Takto můžeme rozdělit týmy podél linií podnikání a specializovat je společně v jednotlivých oblastech. Hlavním problémem tohoto dělení je velmi mnoho kontextů okolo každého takového týmu, což může komplikovat jejich vývojovou činnost. Velké korporace, jako je třeba Amazon, zastávají názor, že tým by měl vlastnit svůj produkt a to po celou dobu života produktu.



Obrázek 2.3: Mikro webové služby - organizace týmu specialistů

Kapitola 3

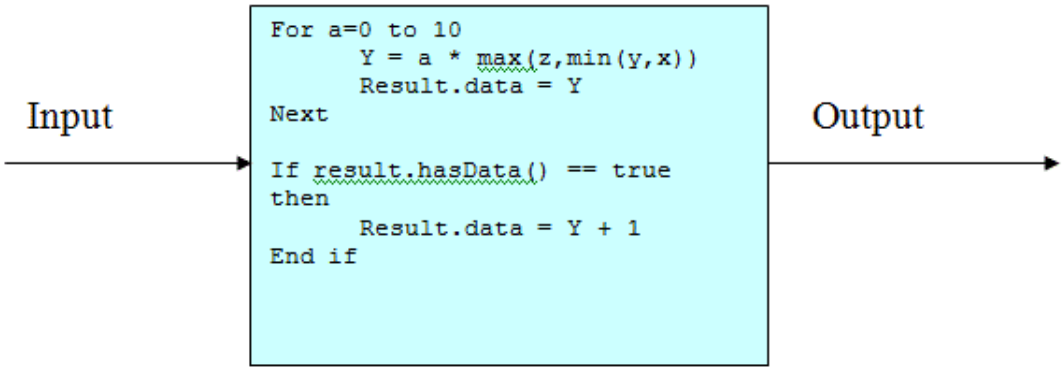
Testování software

3.1 Pohled na systém

Tato kapitola čerpá z knih [4], [2] a [5]. Než budeme pokračovat, rozlišíme si trojici pohledů na systém, které nám říkají, co o testovaném systému víme. Na základě těchto znalostí můžeme zvolit patřičnou úroveň testování.

3.1.1 Bílá skříňka

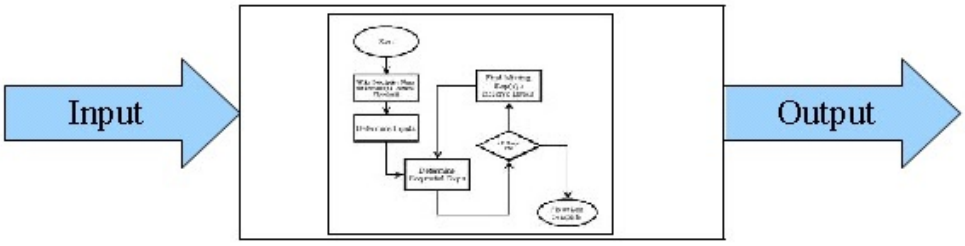
Bílá skříňka (z anglického *White-Box* a někdy také *Glass-Box*) označuje pohled na systém zevnitř. Víme nejen, jak systém pracuje, ale známe i jeho návrh a implementaci. Tento pohled na systém má běžně jeho vývojář. Bílou skříňku znázorňuje obrázek 3.1.



Obrázek 3.1: Testování z pohledu White-box

3.1.2 Šedá skříňka

Šedá skříňka (z anglického *Grey-Box*) označuje pohled na systém, u kterého známe jeho návrh a jak pracuje. Tento pohled na systém má běžně jeho analytik, který jej navrhuje. Ale také zde můžeme uvést i testera, který v jisté úrovni testování potřebuje minimálně znát implementační rozhraní a návrh databáze. Takovým rozhraním mohou být webové služby, které voláme prostřednictvím koncového bodu s parametry. Zde je nutné znát význam takových parametrů a jejich použití. Šedou skříňku znázorňuje obrázek 3.2.



Obrázek 3.2: Testování z pohledu Grey-box

3.1.3 Černá skříňka

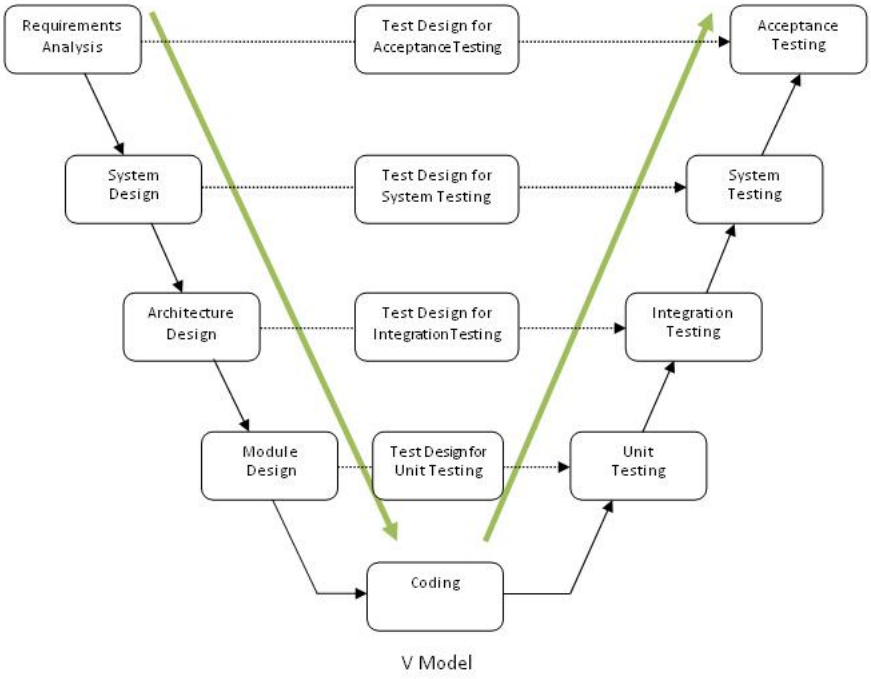
Černá skříňka (z anglického *Black-Box*) označuje pohled na systém, u kterého známe pouze, jak funguje na základě specifikací a vlastním experimentováním. Na základě vstupů nám takový systém poskytuje očekávané výstupy. Černou skříňku znázorňuje obrázek 3.3.

Obrázek 3.3: Testování z pohledu Black-box

3.2 Úrovně testování

Již víme, jakým způsobem můžeme pohlížet na systém a na základě toho systémy umíme dělit do tří kategorií. Na základě znalostí o systému se již můžeme rozhodovat, kterou úroveň testování použít. Jednotlivé úrovně testování nám rozlišuje obrázek 3.4 pro V-model. Každá úroveň testování se váže k příslušné úrovni návrhu, kterou verifikuje:

- Návrh modulů verifikujeme pomocí jednotkového testování.
- Návrh architektury verifikujeme pomocí integračního testování.
- Návrh systému verifikujeme pomocí systémového testování.
- Požadavky na analýzu verifikujeme pomocí akceptačního testování.



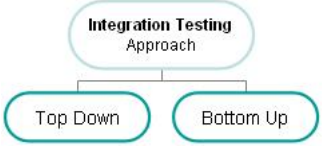
Obrázek 3.4: V-model

3.2.1 Jednotkové testování

Jednotkové testování (z anglického *Unit testing*) řadíme z pohledu systému do kategorie bílé skříňky, tudíž se takovým testováním zábývá pouze vývojář, který se orientuje přímo ve zdrojovém kódu systému. Za jednotku označujeme komponentu, z toho důvodu se můžeme setkat s obdobným názvem jako testování komponent (z anglického *Component testing*). Takovou komponentou může být typicky třída nebo i samotná metoda. Každý jednotkový test by měl být nezávislý na ostátních. Toho lze docílit například vytvořením pomocného objektu (z anglického *mock object*), který simuluje předpokládaný kontext, ve kterém testovaná část pracuje. Jednotkové testování zpravidla generujeme automaticky pomocnými nástroji většinou zabudovanými přímo do vývojového prostředí.

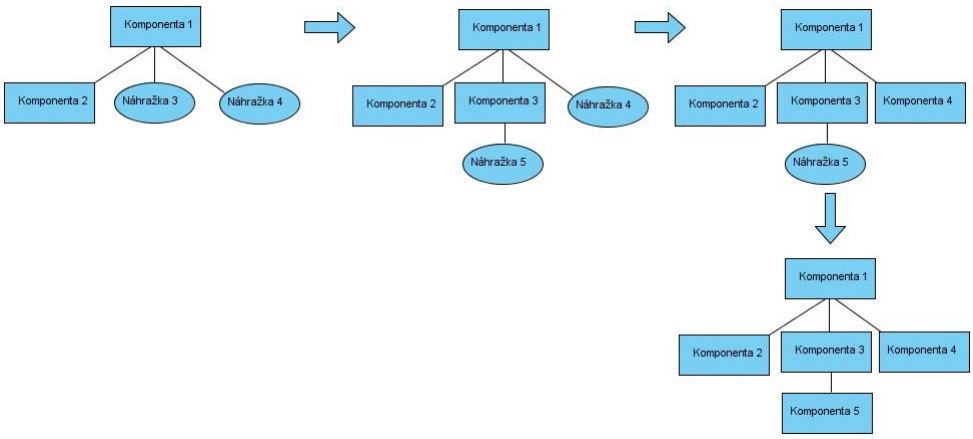
3.2.2 Integrační testování

Integrační testování (z anglického *Integration testing*) řadíme z pohledu systému do kategorie šedé skříňky, protože zde už je zbytečné znát implementaci z pohledu vývojáře. Stačí nám znát na základě návrhu rozhraní jednotlivých komponent. Zde jsme tedy už o úroveň výše a v rámci integřačního testování skládáme jednotlivé komponenty na sebe a testujeme jejich rozhraní. K takovému testování existují dva přístupy 3.5.



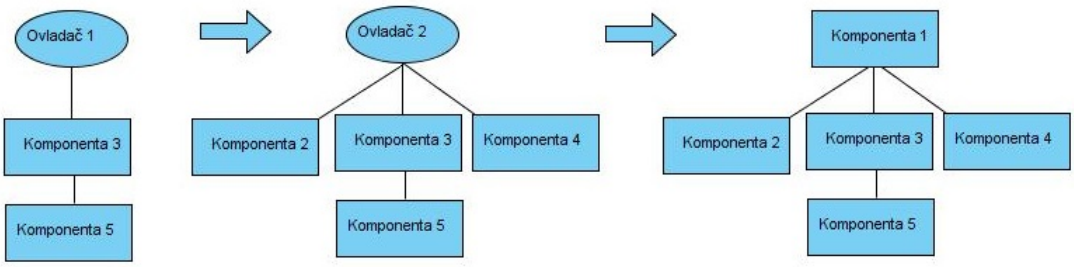
Obrázek 3.5: Přístupy integračního testování

Prvním z přístupů je integrační testování shora dolů (z anglického *top-down testing*). Testováním shora dolů postupně přidáváme jednotlivé komponenty systému od kořene hiearchie. Aby test mohl proběhnout v každé fázi integračního testování, musíme něčím nahradit chybějící komponenty. Nahrazením komponenty vzniká náhražka (z anglického *stubs*). Náhražky musí být jednoduché, ale dostatečné k testování. Její realizace může představovat například zobrazení nějaké zprávy. Tento přístup integračního testování si můžeme ilustrovat obrázkem 3.6, kde je viditelné, že nám v první fázi chybí dvě komponenty, které je třeba nahradit náhražkami, aby bylo možné otestovat komponentu číslo 1. Postupnou integrací následně náhražky nahrazujeme již existujícími komponentami a tento proces opakujeme až do poslední potřebné náhražky.



Obrázek 3.6: Integrační testování shora dolů

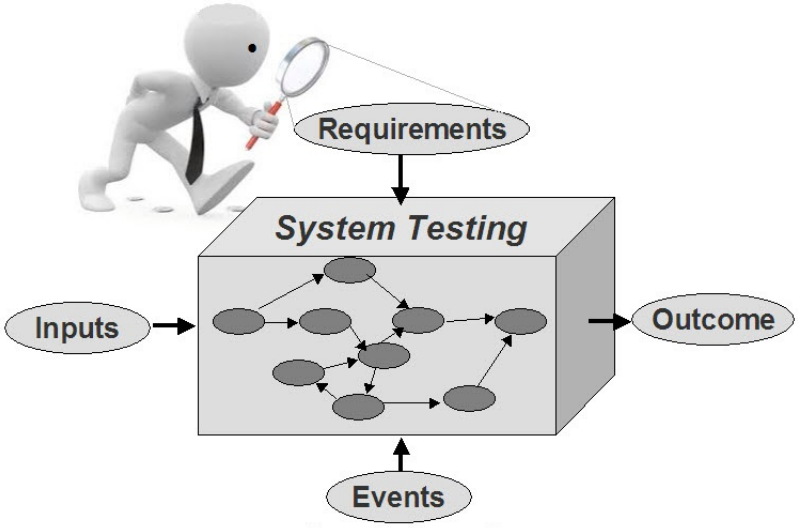
Dalším z přístupů je integrační testování zdola nahoru (z anglického *bottom-up testing*), které probíhá v rámci hiearchie komponent opačným způsobem. Nejdříve testujeme listy, postupně se dostáváme ke kořenům. V tomto případě nahrazujeme chybějící komponenty dočasnou řídící komponentou nazývající se ovladač (z anglického *driver*). Tento přístup ilustruje obrázek 3.7, kde je možné vidět začátek od kompent umístěných na nejnižší úrovni řízené ovladačem na nejvyšší úrovni. Postupnou integrací k takovému ovladači připojujeme postupně další komponenty, které jakmile budou kompletní, nahrazujeme vždy i ovladač komponentou na vyšší úrovni.



Obrázek 3.7: Integrační testování zdola nahoru

3.2.3 Systémové testování

Jakmile systém postupně poskládáme pomocí integračních testů, přichází na řadu systémové testování (z anglického *System testing* nebo také jako *System integration testing*), které aplikaci ověřuje jako funkční celek. Systémové testování zahrnuje přípravu scénářů simulujících různé stavy, které v praxi mohou nastat z pohledu zákazníka. Ověřování provádíme v několika etapách. Nalezené a opravené chyby opět průběžně testujeme. Systémové testování provádíme jako poslední úroveň testů před předáním hotového produktu zákazníkovi. Jednotlivé testovací scénáře bychom měli začít tvořit již od začátku vývoje systému, neboť bez této fáze testování systému ostatní testy postrádají význam a bezporuchovost systému by tak mohla být významně ohrožena. Na obrázku 3.8 můžeme vidět kompletní systém, do kterého na základě zvoleného testovacího scénáře přichází postupně vstupy a události, které jsou zpracovány implementací vypracované dle specifikací. Následně u testovaného systému ověřujeme očekávané výstupy dle zvoleného testovacího scénáře.



Obrázek 3.8: Systémové testování

3.2.4 Akceptační testování

Jakmile předchozí úrovně testování proběhnou bez větších nedostatků, dostáváme se k poslednímu kroku a to k akceptačním testům zákazníka, který dle stanovených scénářů pro jednotlivé podnikatelské procesy ověřuje, zda zákoazníkovi výsledný produkt vyhovuje a příjme ho nebo zažádá o přepracování. Akceptační testy běžně probíhají nad systémem nasazeným na předprodukčním prostředí, na kterém se výsledný produkt vylaďuje, než se z takového prostředí stane produkční, na kterém výsledná aplikace běží v plném provozu.

3.3 Testování webových služeb

Tato sekce čerpá z knih [11] a [4]. Zde se budeme zabývat testováním z pozice klienta, který bude zastupovat uživatelské rozhraní nebo jiný systém, který prostřednictvím webových služeb komunikuje s námi testovaným systémem. Webové služby takového systému mají své rozhraní. V tomto rozhraní nás bude zajímat koncový bod, což je nějaká URL adresa

identifikující webovou službu v síti a dále obsahující parametry. V tomto případě se univerzálně využije testování obou nejznámějších případů webových služeb SOAP i REST, neboť zde budeme hledat hraniční podmínky jednotlivých parametrů.

3.3.1 API

Rozhraní výše uvedených webových služeb nazveme API (z anglického *Application Programing Interface*). API je kolekcí funkcí, které mohou být prováděny jinou softwarovou aplikací nebo komponentou. Koncový uživatel většinou neví, že nějaké API existuje, i přestože je uživatelským rozhraním využíváno. Každé API volání má definovaný počet parametrů, jenž přijímají rozdílné vstupy. Každá varianta vrací rozdílný výsledek.

3.3.2 Testování s využitím analýzy hraničních hodnot

S využitím techniky analýzy hraničních hodnot (z anglického *Boundary value analysis*) v rámci dělení vstupů na třídy ekvivalence (z anglického *Equivalence class partitioning*) se k API nabízí dva přístupy testování:

1. Testování jednotlivých parametrů, což je jednodušší varianta.
2. Testování parametrů, jenž spolu souvisí, což je o něco komplikovanější varianta, protože nám přináší mnohem více možností, které je nutné testem pokrýt. Testujeme takto jednotlivé vzory, jenž identifikují jedinečné kombinace vstupů. Takové testování zahrnuje i nepovinné parametry.

Každý výše uvedený přístup zahrnuje pro každou variantu další varianty, jenž nazýváme hraničními hodnotami. Hraniční hodnoty zahrnují validní i nevalidní vstupy, se kterými je v testu třeba počítat. K dvojici hraničních hodnot libovolného intervalu se ke každé takové hodnotě váží dvě její sousední hodnoty. Označme hraniční hodnotu za x, pak dostáváme posloupnost x - 1, x, x + 1. Níže uvedený příklad vysvětluje hraniční hodnoty pro funkci DIV (děleno).

Příklad: Mějme funkci dělení *DIV(x, y)*, kde x je dělenec a y je dělitel. Hodnoty x i y nabývají hodnot v rozsahu `<-10000; 10000>` a zároveň y nesmí být 0. Následující tabulka 3.1 představuje analýzu hraničních hodnot pro parametr y, jenž následně zahrnujeme do testovacích scénářů.

Ekvivalenční třída	**Hodnoty spodní hranice**	**Hodnoty horní hranice**
`10000 <= y <= 10000`	-10001, **-10000**, **-9999**	**9999**, **10000**, 10001
y = 0	**-1**, 0, **1**	**-1**, 0, **1**

Tabulka 3.1: Analýza hraničních hodnot parametru y

Tučně jsou zvýrazněny validní hodnoty, kterých je 8 a zbývající 4 nevalidní.

Kapitola 4

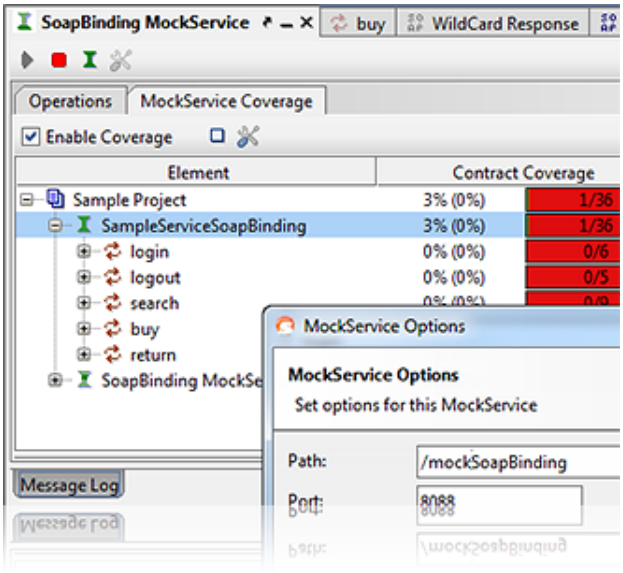
Nástroje automatického testování

4.1 Soap UI

Podle zdroje [10] nástroj Soap UI je otevřený a volně dostupný. Tento nástroj se zaměruje především na testování webových služeb. V rámci úrovně testování bychom ho řádíme mezi integrační a systémové testy. Tento nástroj vlastní jednoduché uživatelské rozhraní, které nám usnadňuje vytvořit testový scénář, který můžeme spouštět automaticky. Takový scénář má postupně vykonávanou stromovou strukturu, kde uzel nejvyšší urovně je jmenný prostor, který obsahuje projekty. Projekt obsahuje testové balíky. Testový balík zahrnuje jednotlivé testové případy a až testové případy obsahují listy, kterými jsou testové kroky. Každý uzel může vykonávat dva skripty. Jeden skript se provádí před provedením potomků (*precondition*) danného uzlu a druhý skript se vykoná po provedení tohoto uzlu (*post condition*). Jednotlivé kroky pak mohou být:

- SOAP požadavek
- REST požadavek
- HTTP požadavek
- AMF požadavek
- Skript
- Volání jiného uzlu
- ...

Pro každý druh požadavku lze nastavit obě jeho části header i body. Každý uzel i list navíc obsahují nastavitelné proměnné. Tedy každý takový krok lze parametrizovat. Soap UI také umí generovat mock services, které lze použít, pokud testovaný systém posílá vlastní volání a potřebujeme mu během testu podstrčit vhodná data. Sílu tohoto nástroje umocňuje skriptování jazykem Groovy, kterým můžeme získat, nastavit či provést, kterýkoliv uzel či list celého testového projektu. Správnou práci systému tak snadno můžeme ověřit přímo proti datům v databázi. Jak takový nástroj vypadá ilustruje obrázek 4.1.



Obrázek 4.1: Nástroj Soap UI

4.2 Selenium

Podle zdroje [9] je Selenium sada nástrojů speciálně zaměřených pro automatizaci webových prohlížečů. Řadíme ho tedy mezi systémové a akceptační testy v rámci úrovně testování. Tento nástroj umí ovládat pomocí skriptu webovou stránku. Takové ovládání zahrnuje množinu operací nad webovým prohlížecem, které zvládnou kterýkoliv uživatelský úkon od otevření stránky přes vyvolávání událostí na stránce, jako je třeba kliknutí na libovolný objekt až po napsání textu do formuláře či interpretaci JavaScriptového kódu. Selenium ovládá instanci webového prohlížečem prostřednictvím jeho nativních funkcí, tedy akce interpretuje pomocí JavaScriptu. Tím lze Selenium aplikovat, na kterýkoliv prohlížeč a kteroukoliv platformu od PC až mobilní zařízení. Testový kód můžeme psát prakticky, v kterémkoliv běžném programovacím jazyce jako PHP, Python nebo Java. Pomocí těchto programovacích jazyků můžeme zároveň volat webové služby, jejichž výsledky můžeme ověřovat přímo na uživatelském rozhraní pomocí nástroje Selenium. K elementům webové stránky přistupujeme pomocí selektorů, jako jsou:

- XPATH
- ID
- NAME
- CSS

4.3 Sikuli

Podle zdroje [8] je Sikuli nástroj automatického testování, který své objekty rozpoznává a identifikuje pomocí obrazových vzorů na uživatelském rozhraní. Podobně jako Selenium i Sikuli řadíme mezi akceptační a systémové testy. Ale narozdíl od nástroje Selenium se

neomezujeme pouze na webový prohlížeč, ale na vše, co můžeme vidět na monitoru. Sikuli vlastní jednoduché uživatelské rozhraní, kde jednotlivým obrázkům, které nátroj hledá na obrazovce, přiřazujeme akce. Podobně jako v nástroji Selenium funkce těchto akcí jsou přizpůsobeny pro vyvolávání události na uživatelském rozhraní či čtení nebo zadávání textu. V Sikuli se píší skripty pomocí jazyka Jython, což je jazyk Python schopný pracovat s Java knihovnami. Podobně tedy jako u nástroje Selenium můžeme pomocí skriptu volat webové služby a následně je ověřovat přímo na uživatelském rozhraní, které v tomto případě může být jak webového, tak i desktopového typu.

Kapitola 5

Závěr

Cílem této práce bylo vysvětlit možnosti testování webových služeb. V první kapitole jsme se seznámili s datovým nosičem webových služeb. Vysvětlili jsme si tedy strukturu XML a kde ho lze aplikovat. Na okraj jsme si ukázali, jak lze XML uspořádat a formátovat pomocí XSL a XSLT. Tím jsme byli schopni porozumět XML jako datovému nosiči webových služeb. Dále jsme se již dozvěděli základní informace o webových službách. Známy jsou nám přístupy a protokoly pro jejich implementaci, čímž jsme si dále mohli objasnit i pojmy, jako jsou SOAP, WSDL, REST. Na okraj jsme se seznámili s novinkou, kterou jsou mikroslužby.

V následující kapitole jsme si ukázali pohledy na systém a dále jsme si vymezili úrovně testování. Zjistili jsme, že na webové služby jsou vhodné integrační testy pro testování jejich rozhraní. Ohledně testování takového rozhraní jsme si ukázali příklad ohledně analýzy hraničních hodnot na ekvivalenčních třídách parametrů, kde jsme se dozvěděli, jak určovat validní a nevalidní testové případy.

V případě testování software si ukážeme pohledy na systém pro vymezení a lepší pochopení volby vhodné úrovně testování pro případ pokrytí webových služeb testem. Dále využijeme znalostí o webových službách a základních znalostech o testování a přistoupíme k samotnému testování webových služeb s využitím analýzy hraničních hodnot na ekvivalenčních třídách parametrů.

Na závěr jsme si představili vhodné nástroje automatického testování pro pokrytí webových služeb automatickým testem. Představili jsme si nástroje automatického testování Soap UI, Selenium a Sikuli. Zařadili jsme je k patřičným úrovním prováděných testů a objasnili, pro které případy jsou jednotlivé nástroje užitečné pro pokrytí automatickým testem.

Literatura

[1] Bradley, N.: *XML kompletní průvodce*. Grada Publishing, 2000, iSBN 80-7169-949-7.

[2] Burnstein, I.: *Practical Software Testing*. Springer-Verlags, 2002, iSBN 0-387-95131-8.

[3] Butek, R.: Which style of WSDL should I use? [online]. `http://www.ibm.com/developerworks/webservices/library/ws-whichwsdl/`, [cit. 2015-07-21].

[4] Farrell-Vinay, P.: *Manage Software Testing*. Aurbach Publications, 2008, iSBN 978-0-8493-9383-9.

[5] Glenford J. Myers, T. B., Corey Sandler: *The Art of Software Testing*. Wiley, 2011, iSBN 978-1118031964.

[6] James Lewis, M. F.: Microservices [online]. `http://martinfowler.com/articles/microservices.html`, [cit. 2015-07-21].

[7] Kolektiv autorů: Service-Oriented Architecture (SOA) Definition [online]. `http://www.service-architecture.com/articles/web-services/service-oriented_architecture_soa_definition.html`, [cit. 2015-07-21].

[8] Kolektiv autorů: Sikuli [online]. `http://www.sikuli.org/`, [cit. 2015-07-21].

[9] Kolektiv autorů: What is Selenium [online]. `http://docs.seleniumhq.org/`, [cit. 2015-07-21].

[10] Kolektiv autorů: What is Soap UI [online]. `http://www.soapui.org/about-soapui/what-is-soapui-.html`, [cit. 2015-07-21].

[11] Lisa Crispin, J. G.: *Agile Testing: A Practical Guide for Testers and Agile Teams*. Addison-Wesley Professional, 2009, iSBN 978-0321534460.

[12] Paul Ammann, J. O.: *Introduction Software Testing*. Cambridge University Press, 2008, iSBN 978-0-521-88038-1.

[13] Rodriguez, A.: RESTful Web services: The basics [online]. `http://www.ibm.com/developerworks/webservices/library/ws-restful/`, [cit. 2015-07-21].

[14] Snell, J.: Call SOAP Web services with Ajax [online]. `http://www.ibm.com/developerworks/webservices/library/ws-wsajax/`, [cit. 2015-07-21].