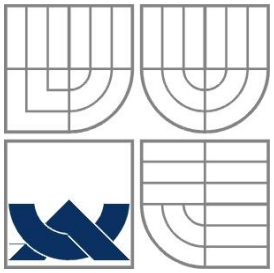
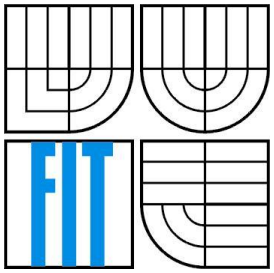


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

AUTOMATIZOVANÉ TESTOVÁNÍ WEBOVÝCH STRÁNEK
AUTOMATED TESTING OF WEB PAGES

BAKALÁŘSKÁ PRÁCE
BACHELOR THESIS

AUTOR PRÁCE — Vilém Jeniš
AUTHOR

VEDOUCÍ PRÁCE — Ing. Marcela Šimková
SUPERVISOR

BRNO 2014

Abstrakt

Práce se týká automatizace testů uživatelského rozhraní. Po stručném úvodu jsou popsány některé nástroje, které usnadňují tvorbu automatických testů uživatelského rozhraní webových aplikací a následně je prezentován návrh architektury automatizačního frameworku, který umožňuje jeho uživatelů psát automatizační testy jednoduše. Práce uzavírá rozborem praktických příkladů a validací návrhu.

Abstract

This paper deals with user interface testing automation. After a brief introduction follows a description of tools facilitating the making of a suite of automated web application user interface tests. Next is a proposition of an architecture design of an automation framework allowing its users to write user interface test easily. The paper closes with an analysis of practical examples and validation of the architecture design.

Klíčová slova

Testování, automatizace, Selenium, .NET, NUnit

Keywords

Testing, automation, Selenium, .NET, Nunit

Citace

Jeniš Vilém: Automatizované testování webových stránek, bakalářská práce, Brno, FIT VUT v Brně, 2014

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Marcely Šimkové. Další informace mi poskytl Mgr. Michal Pietrik. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Vilém Jeniš
20.5.2014

Poděkování

Rád bych poděkoval paní inženýrce Šimkové za ohromnou trpělivost a pečlivost při vedení této práce, stejně tak jako magistru Pietrikovi za sdílení svých zkušeností a poskytnutí pramenů a informací pro tuto práci.

Obsah

1 Úvod

Software je velmi rychle rostoucí průmysl a s agilními metodikami dosahuje nevídaného růstu. Každá společnost produkující software se snaží držet tento svižný krok. Přesto však má-li být software v produkci k něčemu dobrý, musí být řádně otestován a nalezené chyby opraveny. To činí z testování důležitou součást vývojové metodiky ať už je agilní nebo ne. Testováním ovšem nelze dosáhnout stoprocentně odladěného produktu, nicméně lze jím pokrýt jistou množinu scénářů a možných chyb. Čím větší je pak tato množina, tím méně chyb pak čeká na to, aby byly odhaleny zákazníky. Hned několik technik bylo vyvinuto, aby mohla být tato množina co největší. Čím dříve ve vývojovém cyklu chyba odhalena a opravena, tím menší je její finanční dopad.

Agilní metodika vývoje zvaná SCRUM definuje akceptační kritéria nebo také akceptační testy[11]. Jedná se o test sepsaný z pohledu uživatele formou scénáře, který popisuje jednotlivé kroky. Je-li možné všechny kroky úspěšně vykonat, znamená to, že testovaná funkcionalita správně naimplementována a splňuje základní požadavky. Tyto testy nemusí být ovšem vždy jednoznačné, o tom, jestli byl test úspěšně vykonán, rozhoduje ten, kdo scénář prochází a hraje tedy roli také lidský faktor a subjektivní pohled na věc. Zvětšuje se tím ovšem množina pokrytých scénářů bez nějakého nároku na architekturu implementace (testovatelnost).

Test driven development – metodika jejímž jádrem je, že testy na funkcionalitu se píší ještě předtím, než je funkcionalita implementována. Test je v ideálním případě automatizován a tak může být spuštěn a vykonán kdykoliv, je-li upravován kód, který je testem pokrýván. Test je napsán takovým způsobem, aby zajišťoval, správné chování v nejčastějších scénářích. Výhodou automatizovaných testů je, že jsou jednoznačně definované. Na druhou stranu kód musí být psán s ohledem na to, aby byl testovatelný. To není vždy jednoduché a samotný počáteční návrh architektury musí toto zohledňovat.

Zaměříme se spíše na nový trend, který pomáhá odhalovat chyby v softwaru a tím i vylepšovat jej a zároveň usnadňuje práci inženýrům z oddělení zajišťování jakosti (testerům). Budeme mluvit o automatizaci testů uživatelského rozhraní. Konkrétněji nám půjde o automatizaci uživatelského rozhraní webových aplikací. Budeme rozebírat prostředky a nástroje pro vývoj takových testů, použitelnost a udržovatelnost a také přínos a užitek v porovnání s cenou a časem stráveným vývojem. Vytvoříme ucelený návrh architektury frameworku pro psaní automatických testů UI s velkým důrazem na snadné psaní samotných testů a udržovatelnost celé sady. Následně jej za pomocí vybraných nástrojů jej naimplementujeme. Nezapomeneme ovšem nejdříve zdůraznit nutnost jiných druhů automatizovaných testů stejně tak jako lidských testerů.

Následující kapitola pojednává obecněji o automatizaci testování, abychom zavedli kontext nutný pro následný podrobný rozbor hlavního tématu.

2 Automatizované testy

Automatizace testování je žhavé téma poledních několik let a zájem o něj neklesá zejména proto, že odpověď na otázku zda automatizovat není přímočará. Musí se zvážit několik aspektů vyvíjeného produktu. Je nutné znát svůj produkt, aby bylo možné objektivně rozhodnout, zda je vhodné automatizovat. Je vhodné zvážit cenu udržování automatizovaných testů v porovnání s cenou vykonání manuálního testu. Také je potřeba zamyslet se nad cenou toho, když test selže, nebo se objeví chyba v produkci. Zvážíte-li všechna tato fakta, můžete se rozhodnout, zdali se automatizace testování vyplatí v porovnání s cenou a úsilím věnovaným manuálnímu testování. [1] Existuje ovšem hned několik úrovní, na kterých se dají psát automatizované testy. Nyní se podíváme na jejich jednotlivá specifika, klady a zápory.

2.1 Unit testy

Unit testy jsou nejznámějším a nejpoužívanějším typem automatizovaných testů. Unit testy slouží k ověření toho, že izolovaná část kódu funguje správně. Cílem unit testů je mít sadu testů, která testuje funkci systému přesně danými způsoby, ale také poskytuje správné příklady použití a interakce s určitými typy objektů [1]. V příloze 2 je ukázka unit testu, který zjišťuje, zdali objekt typu má `FacebookPostInfo` po vytvoření před uložením do databáze skrze API nulovou hodnotu primárního klíče.

2.1.1 Charakteristiky unit testů

Unit testy jsou nezávislé. To znamená, že nezávisí na konkrétním souboru na konkrétním místě či databázi aby byly úspěšné. Nejsou-li nezávislé, je známo, že to pak znamená hodiny strávené laděním a zkoumáním proč testy selhávají [1].

Unit testy jsou jednoduché, zřetelně čitelné a testují pouze malý úsek kódu. Když unit test selže, je jasné, v jakém úseku kódu je chyba, a co je špatně. Není potřeba ani podrobná zpráva o tom, co bylo špatně. Vše je patrné z názvu testu. Například, název `Test853` na rozdíl od `WhenNewExampleObjectIsCreatedTheCountIsEqualToOne` neprozrazuje mnoho o podstatě testu. [2]

Unit testy jsou rychlé. Tím, že jsou nezávislé, nečekají na odpovědi jiných komponent a také tím, že každý unit test testuje jen malou část kódu, trvání unit testů by nemělo být nijak dlouhé a měly by se spouštět často. Jsou dokonce známy implementované mechanizmy, které zajišťují spuštění unit testů při jakémkoliv pokusu vložit změnu kódu do verzovacího nástroje či jiné správy zdrojového

kódu. Jsou-li unit testy pomalé, demotivuje to vývojáře, aby je spouštěli, a takové testy pak ztrácejí smysl. [2]

2.1.2 Výhody a nevýhody unit testů

Unit testy slouží k tomu, aby odhalily chyby hned v počátku. Poslouží však pouze k odhalení určitého typu chyb. Jde o chyby na nízké úrovni v logice aplikace. Jeden test testuje pouze malý izolovaný kousek kódu a testy jsou na sobě nezávislé. Obrovskou výhodou je rychlost unit testů.

Nevýhodou unit testů je, že ne všechen kód lze testovat jejich pomocí. Architektura projektu se musí navrhovat s úmyslem unit testy používat nebo následně refaktorovat. Každá nová třída, která pak přibývá, musí být dostatečně izolovatelná, aby šla vybavit unit testy.

2.2 Integrační testy

Integrační testy testují spolupráci dvou nebo více komponent. Slouží také k testování za hranicí aplikace. Testují její interakce a integraci s jinými službami a aplikacemi. Cílem integračních testů je tak odhalit špatnou interakci komponent aplikace a chyby v integraci se službami třetích stran, jako nesprávné ošetření neočekávaných stavů, špatnou interakci spolupracujících modulů či nebezpečnou reakci na podvrženou komunikaci. [1] Jako příklad poslouží ověření uživatele v databázi pomocí uložené procedury. Ověříme tím, že aplikace správně komunikuje s databází, že správně z hesla generujeme otisk, který předáváme uložené proceduře a také, že procedura funguje správně a se správným heslem a jménem uživatele ověří a v opačném případě ne.

2.2.1 Charakteristiky integračních testů

Integrační testy mohou být pomalé. Při testování komunikace se službami třetích stran dochází k zasílání dat mezi Vaší aplikací a danou službou. To může trvat nějaký čas a ten je potřeba brát v úvahu. [1]

Integrační testy jsou křehké. To znamená, že mohou selhat z nenadálých či nepředpokládatelných důvodů. Závisí na službách třetích stran. Pro jejich úspěšný průběh je nutné, aby byla služba dostupná a fungovala správně. Dále vyžadují, aby se neměnilo aplikační rozhraní. Vzhledem k unit testům, vyžadují integrační testy odlišný přístup k výsledkům. Zatímco chceme, aby unit testy zůstávaly stále zelené, je možné, že integrační test selže z jiných důvodů, nad kterými nemáte kontrolu. Například že není v tento moment dostupná databáze, protože server je v údržbě.

2.2.2 Výhody a nevýhody integračních testů

Integrační testy, které testují spolupráci několika komponent aplikace, jsou dobré k ověření konzistence návrhu a toho, že komponenty či systémy dobře spolupracují.

Vzhledem k tomu, že integrační testy mohou záviset na službách třetí strany, stává se, že selžou, ačkoliv je vše v pořádku. To ovšem neznamená, že takové testy jsou k ničemu. Když například začnou selhávat testy integrace Twitteru do vaší aplikace, může to být i proto, že síť na, které testujete, nemá přístup do internetu, nebo Twitter pozastavil své služby kvůli údržbě.

2.3 Automatické testy uživatelského rozraní

Automatické testy uživatelského rozhraní (UI) jsou strojem prováděné testy, které se snaží co nejvěrněji napodobit koncového uživatele. Jedná se o testy takzvaně funkcionální, poněvadž testují přímo funkcionalitu tak, jak ji vnímá uživatel. To zapříčiňuje, že testy jsou často křehké, poněvadž funkcionalita se mění. V několika-krokovém konfiguračním dialogu například mohl přibýt nový krok, protože byla naimplementována nová část systému, která je taktéž potřeba nakonfigurovat. Test s tímto krokem nepočítá a selže.

Některé nástroje pro automatizaci uživatelského rozhraní spočívají v tom, že se nahrává a následné přehrává sekvence kliků se souřadnicemi, kde k němu došlo a sekvence stisknutých kláves. Takové testy jsou velmi náchylné a křehké. Stačí, aby se přemístilo například tlačítko pro přihlášení do systému a jakýkoliv test, který pracuje v zabezpečené části systému je k ničemu. Před deseti lety (2004) se objevily první pokročilé nástroje na automatizaci internetových prohlížečů. Tyto nástroje jsou daleko specializovanější a poskytují prostředky pro to, aby testy uživatelského rozhraní webových aplikací byly stabilnější a lépe udržovatelné.

2.3.1 Charakteristiky automatických testů UI

Testy uživatelského rozhraní jsou pomalé. Jsou ještě o stupeň pomalejší než integrační testy. Celý systém musí spolupracovat na všech úrovních a testy se musejí často vypořádat i s delší odezvou systému aby neselhávaly jen proto, že testovaná aplikace neodpovídá dostatečně rychle.

Testy uživatelského rozhraní spoléhají na služby třetích stran. Stejně tak, jako integrační testy, testy uživatelského rozhraní často potřebují k úspěšnému průběhu, aby na správné adrese běžela správná služba ve správné konfiguraci.

Testy UI jsou křehké. Souvisí s tím také předchozí bod. Testy UI vyžadují speciální přístup, aby byly jednoduše udržovatelné. Jinak se může také stát, že sebemenší změna v systému bude znamenat, že se musí přepsat nemalé množství testů.

Testy uživatelského rozhraní vyžadují sadu testovacích dat. Často se stává, že k tomu, aby uživatel mohl plnohodnotně využít určitou funkcionalitu, je potřeba dodat aplikaci nějaká data. Chceme-li například testovat, že pravidelný zákazník e-shopu dostane automaticky slevu 10% na svůj nákup, potřebujeme uživatele s historií a různé produkty, které může přidat do košíku.

2.3.2 Výhody a nevýhody automatických testů UI

Automatické testy uživatelského rozhraní simulují uživatele při provádění reálných scénářů a kontrolují, že scénář je možné bez problémů dokončit a uživatel se tím pádem neocitne v situaci, kdy by neměl možnost dokončit nějakou akci. Výhodou těchto testů je hlavně to, že se testují reálné scénáře a simulací uživatele a jeho chování se výsledky velmi blíží manuálnímu testování. Manuální testování však v žádném případě nenahrazuje. Automatické testy UI jsou testy funkcionální. Testují tedy funkcionalitu. Chceme-li otestovat, že nějaké stránky vypadá vizuálně pěkně, a že je kompatibilní se všemi hlavními prohlížeči, je na místě manuální testování. *„Nechme počítač dělat to, v čem je dobrý a nechme na testerech to, v čem jsou dobří oni.“* [1] Testy uživatelského rozhraní jsou často horkým kandidátem na regresní testy, jedná-li se o chybu v logice aplikace špatně oddělené od prezentační vrstvy, nebo je-li chyba spojená s chováním prohlížeče či jde-li o chybu spolupráce klientské a serverové logiky.

3 Nástroje pro automatizaci internetových prohlížečů

Abychom mohli psát testy, které napodobují uživatele co nejvěrněji, musíme k aplikaci přistupovat stejným způsobem, jako to dělá její uživatel. Jelikož je tato práce zaměřená na testování webových aplikací, prostředkem, který uživatel používá pro interakci s aplikací je internetový prohlížeč. Internetové prohlížeče se dnes počítají na desítky. 90% uživatelů internetu na osobních počítačích však používá jen 3 z nich. //Zdroj!!!

Následující dvě kapitoly představí dva různé přístupy k simulaci uživatele webové aplikace.

3.1 Simulace uživatele bez grafického uživatelského rozhraní

3.1.1 HTMLUnit

HTMLUnit je nástroj pro psaní automatizovaných testů uživatelského rozhraní webových aplikací. Mezi jeho hlavní charakteristiky patří, že je bez grafického uživatelského rozhraní (GUI) a simuluje prohlížeč, aniž by cokoliv zobrazoval. Další důležitou charakteristikou je, že je naprogramován v Javě a je tak vhodný a jednoduše použitelný hlavně pro testování webových aplikací psaných právě v Javě. [6]

Mezi výhody, které z toho vyplývají, patří rychlost a jednoduchost. HTMLUnit má také solidní a stále rostoucí podporu JavaScriptu. Nevýhodou je, že HTMLUnit v žádném případě nesimuluje žádný moderní prohlížeč, jako je Firefox, Chrome, či Internet Explorer. Tyto prohlížeče disponují funkcionalitou pro urychlení načítání, která často zahrnuje pre-fetching, caching a další vychytávky, které jednoduchý klient HTMLUnitu nepodporuje. Samotnou absence GUI také patří mezi nevýhody. Nelze díky tomu jednoduše přehrát testovaný scénář a pozorovat, co se děje na obrazovce uživatele, jehož chování test simuluje.

V žádném případě to neznamená, že HTMLUnit je nepoužitelný. Je velmi vhodný na testování zátěže a výdrže. Zajímá-li nás nejen čas odpovědi na určitý dotaz, ale chceme také kontrolovat obsah odpovědi, popřípadě provádět ve velkém množství paralelně jednoduché scénáře, je HTMLUnit tou správnou volbou pro projekty psané v Javě.

3.2 Řízení prohlížeče

Řízení prohlížeče znamená, že je otevřená fyzická instance prohlížeče a Framework následně využívá vestavěnou podporu pro automatizace daného prohlížeče pro interakci s aplikací.

Existuje několik velkých hráčů na tomto poli. Abychom zavedli nutný kontext, uvedli na příkladech rozdíly a podobnosti a nakonec obhájili výsledný výběr nástroje, představíme si některé z nich.

3.2.1 Telerik Testing Framework

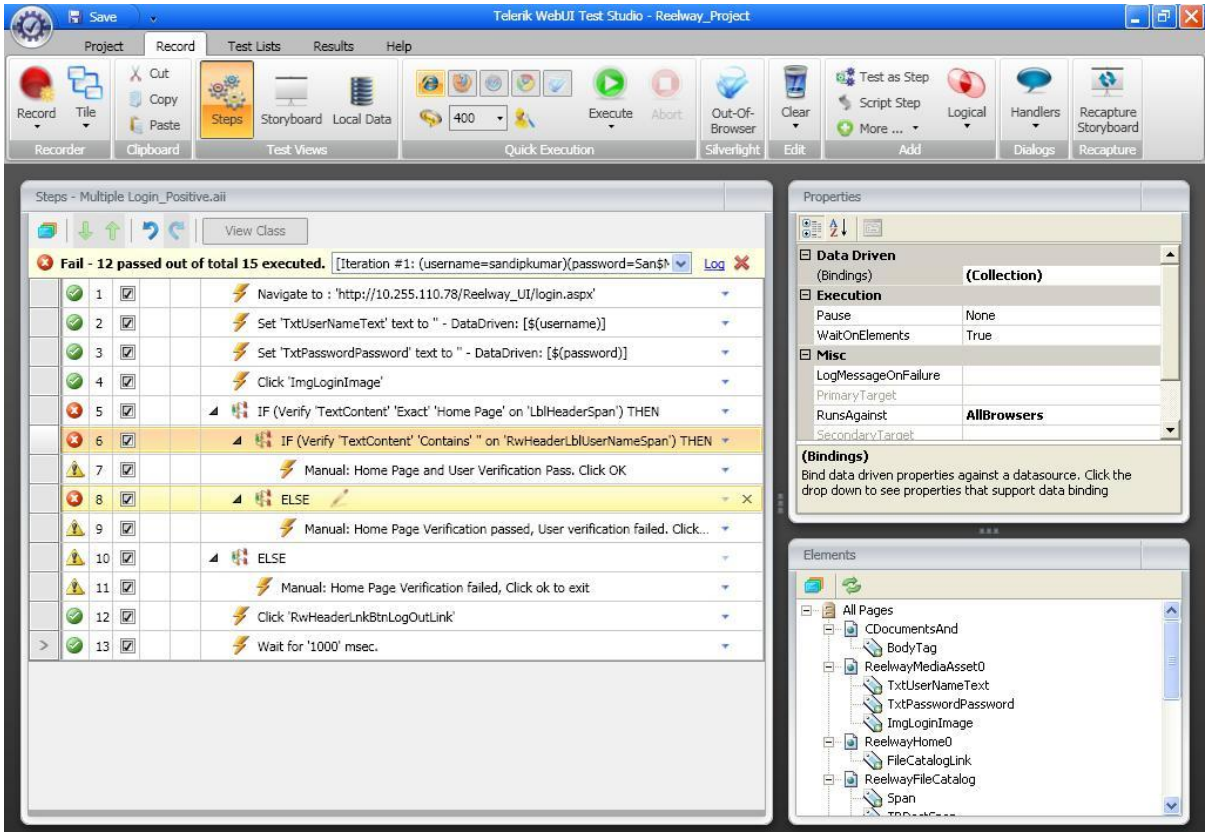
Telerik testing Framework (TTF) je soubor knihoven, který poskytuje velmi bohaté aplikační rozhraní (API) a umožňuje také řídit prohlížeč.

Jednou ze zdůrazňovaných předností je bohaté API. Popravdě řečeno bohaté API může velmi snadno začátečníka odradit, obzvláště, není-li intuitivně strukturované. Další je kompatibilita mezi prohlížeči. Tím je zřejmě myšlena podpora čtyř hlavních prohlížečů: Chrome, Firefox, Safari a Internet Explorer. Telerik dále zdůrazňuje jednoduchou instalaci velmi jednoduché nastavení. Ačkoli je znát, že byla projevena nemalá snaha o to, aby vše bylo jednoduché a snadno použitelné, v porovnání s ostatními nástroji nelze označit použití knihovny od Teleriku jako jednoduché. Velkou předností je však velká podpora HTML5, JavaScriptu, AJAXu a drag&drop operací.

Telerik také inzeruje nativní podporu Telerik uživatelských ovládacích prvků ve svém testovacím frameworku a to ve WPF, Silverlight, AJAX a Kendo UI. To je pravděpodobně příčinou toho, že tento Framework je relativně velký.

V sadě nástrojů Telerik je také nástroj pro nahrání a přehrávání skriptů. Testy takto vytvořené jsou ale upravovatelné pouze v programu, který je součástí Telerik Test Studio. Díky nástroji se podařilo eliminovat špatnou čitelnost testů, nicméně testy stále trpí špatnou udržovatelností. Testy

mají formát seznamu kroků, které se postupně vykonávají. Obrázek 4.1 ilustruje, jak takový test vypadá.



3.1 - Ukáza testu v Telerik Test studiu

3.2.2 Sahi

Sahi vzniklo v roce 2005 jako open source projekt s konkrétním zaměřením na automatizaci právě vznikajícího Webu 2.0 jakožto nástroj určený pro testery. V roce 2010 vznikla placená verze Sahi, protože firmy používající tento nástroj prahly po firemní podpoře a finanční podpora zase na druhou stranu umožnila tomuto nástroji daleko větší rozlet a rozvoj.

Sahi umožňuje také nahrávat a přehrávat skripty. Toto je jeho hlavní devizou. Jelikož je tento

```
_navigateTo("http://sahi.co.in/demo/training/");
_setValue(_textbox("user"), "test");
_setValue(_password("password"), "secret");
_click(_submit("Login"));
_setValue(_textbox("q"), "3");
_setValue(_textbox("q[1]"), "1");
_setValue(_textbox("q[2]"), "2");
_click(_button("Add"));
_assertEqual("1800", _getValue(_textbox("total")));
_click(_button("Logout"));
```

4.2 - Ukázka skriptu nahraného v Sahi

nástroj mířen právě na testery. Skripty jsou samozřejmě křehké, ale pomohou brilantně a hlavně rychle s repetitivními úkoly. Skripty lze přehrávat ve čtyřech prohlížečích. Sahi umožňuje ovšem také skripty dekomponovat, psát vlastní funkce a do určité míry strukturalizovat kódový základ automatizačních testů. Stále se však jedná jen o strukturovaný kód bez jakéhokoliv objektového modelu. Ukázka výsledného skriptu je v ukázce kódu 4.2.

3.2.3 Geb

Geb je zdarma a má podobnou filozofii jako Sahi. Automatizované scénáře se také píšou v interpretovaném skriptovacím jazyce, který klade důraz na čitelnost. Jazyk je podobný JavaScriptu už jenom proto, že s DOM stromem se pracuje podobně jako v jQuery. Je k dispozici tedy jakýsi objektový model, ale právě kvůli jQuery práci s DOM je při zachování filozofie jen velmi těžko rozšiřitelný nad úroveň klasických HTML prvků.

Geb nepodporuje nahrávání skriptů, nicméně tyto nástroje nejsou tak jako tak vhodné pro tvorbu propracované a udržované sady testů UI. V ukázce 4.3 je ilustrace takto psaného kódu, který navštíví webovou aplikaci, zkontroluje nadpis, zadá přihlašovací údaje, potvrdí formulář a zkontroluje nadpis.

```
import geb.Browser

Browser.drive {
    go "http://myapp.com/login"

    assert $("h1").text() == "Please Login"

    $("form.login").with {
        username = "admin"
        password = "password"
        login().click()
    }

    assert $("h1").text() == "Admin Section"
}
```

4.3 - Ukázka skriptu pro Geb

3.2.4 WatiN

WatiN je velkým hráčem na poli automatizace internetových prohlížečů. Název i knihovna samotná je inspirována knihovnou WatiR pro Ruby. WatiN znamená Web application testing in .Net.

WatiN je open-source C# knihovna pro řízení internetových prohlížečů, která v základu podporuje Internet Explorer a Firefox. Jednoduše si poradí se základními HTML prvky ale je přímo stavěný na rozšiřování. Přirozeně podporuje klasické schéma stránek a ovládacích prvků (Pages and Controls model), které je základem pro vývoj webových aplikací ve WebForms. Jak pak vypadá

rozšíření, je popsáno a ukázáno přímo na stránkách WatiNu: http://watinandmore.blogspot.cz/2009/12/wrapping-complex-logic-in-control.html Prakticky jde o to, že není problém podědit z jednoduchého ovládacího prvku a rozšířit jeho možnosti nebo například podědit z obecného ovládacího prvku a vytvořit prvek zcela nový, který se bude skládat z několika jednodušších. Přesně tak, jak funguje toto schéma ve WebForms.

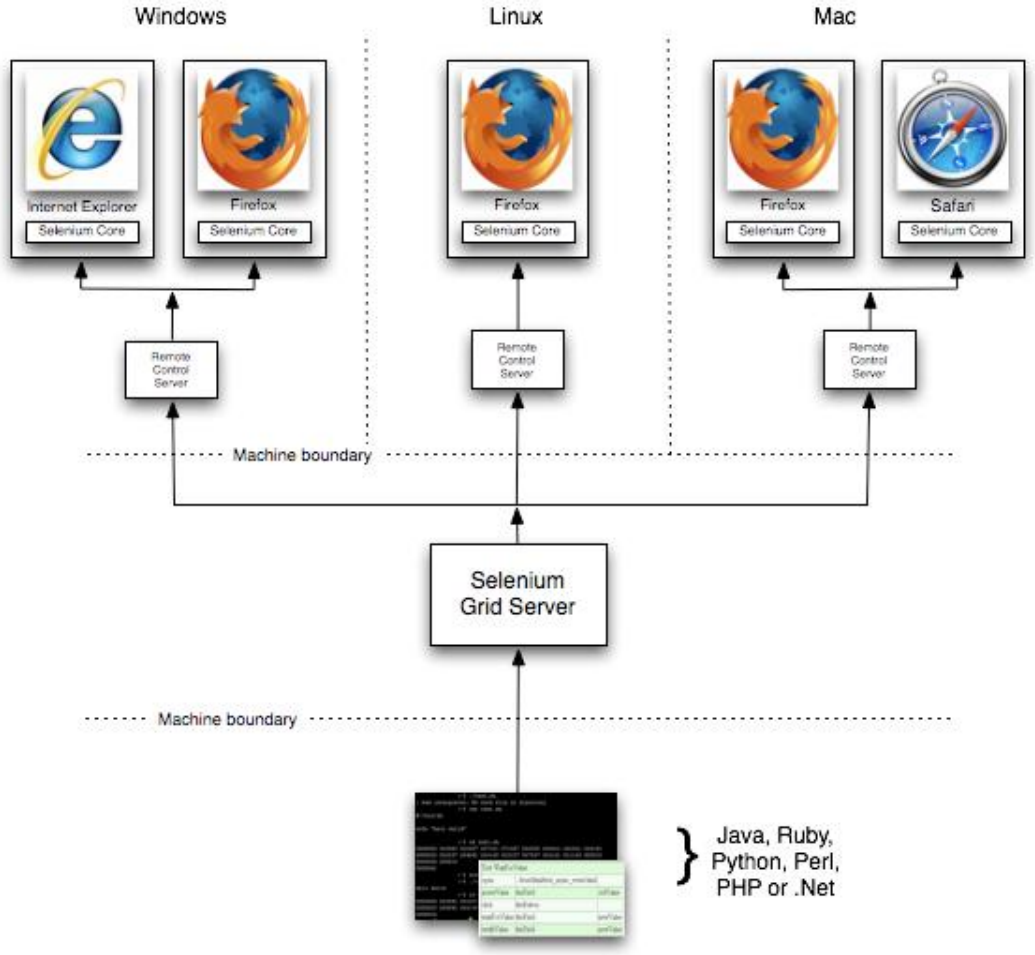
WatiN nedisponuje prostředkem pro nahrávání a přehrávání scénářů. To však není na škodu. Jelikož jsou kvalitně navržené a konstruované testy daleko lépe udržovatelné, je menší velikost a jednoduchost produktu WatiN výhodou.

3.2.5 Selenium

Selenium je jedním z prvních open-source projektů zaměřených na řízení internetových prohlížečů. Vznikl v roce 2004, tedy v době, kdy na trhu bylo jen velmi málo nástrojů na nahrávání a přehrávání scénářů a dva největší: Quick Test Pro a Test Director byly vyvíjeny firmou Mercury a byly velmi drahé. Selenium je protilátkou proti otravou rtutí (anglicky mercury) ale také reakcí na předražené nástroje od firmy Mercury.

Knihovna pro řízení prohlížečů, kterou Selenium obsahuje, poskytuje podobné možnosti, jako WatiN. Rozšiřování jednotlivých prvků pomocí dědičnosti, či skládání dohromady složitějších prvků z těch jednoduchých jsou přirozeným přístupem při stavění testů.

Selenium suite je v dnešní době plnohodnotnou sadou nástrojů na řízení prohlížečů. Poskytuje prostředky pro nahrávání a přehrávání scénářů a knihovnu pro řízení prohlížečů pro Javu, Python, Ruby a v neposlední řadě také C#.Net. Knihovnu lze však stáhnout zcela odděleně od jakékoliv jiné knihovny, nebo nahrávacího nástroje. V základu selenium umí řídit Internet Explorer a Firefox stejně jako WatiN, nicméně existují ‘drivery‘ třetích stran pro Operu, Chrome, a dalších 6 prohlížečů z velké části pro mobilní platformu. Selenium Suite obsahuje také prostředky pro distribuci testů na několik počítačů zvané Selenium Grid. Grid tak dovoluje testování několika scénářů paralelně na několika strojích v několika prohlížečích na několika operačních systémech. Obrázek 4.4 je ukázkové schéma použití Selenium Gridu.



Obrázek 3.4 - Schéma použití Selenium Gridu

Nesmírná síla, možnosti a flexibility Selenia z něj činí volbu nástroje pro řízení internetových prohlížečů pro tuto práci. Možnost automatizovat scénáře i na mobilních prohlížečích, paralelizovat jejich přehrávání a definovat si vlastní ovládací prvky, ale hlavně také neustálý růst knihovny a dalších možností zaručují dostatečnost a aktuálnost nástroje pro jakékoliv scénáře.

4 Nástroje pro správu, organizaci a spouštění automatických testů

Našim cílem je automatizovat testování. Pro tyto účely existuje také celá řada více či méně vhodných nástrojů. Nástroj pro správu testů budeme používat proto, že cílem je vytvořit rozsáhlou knihovnu automatizovaných scénářů a nejen malou množinu tzv. ‘smoke testů‘, které ověří pouze, že

aplikace není nijak zásadně nefunkční. Aby bylo možné testy organizovat a selektivně spouštět efektivním způsobem, je takovýto nástroj nutností.

Když se bavíme o takovýchto nástrojích, jedná se většinou o nástroje uzpůsobené pro unit testování. Některé však šly dále, a ačkoliv jejich primární určení a misí zůstává správa unit-testů, poskytují dostatečnou volnost a možnosti pro správu integračních i automatizovaných testů UI. Následuje rozbor dvou knihoven, z nichž jedna je zástupcem knihoven čistě pro tvorbu unit-testů a tuto svoji funkci plní skvěle a druhá poskytuje ohromnou flexibilitu a možnosti, které z ní činí skvělý nástroj pro správu jakékoliv sady testů.

4.1 MS Test

MS Test je nástroj pro spouštění unit-testů vytvořených ve Visual Studio Unit Testing Frameworku. Testy v něm psané mají klasickou architekturu: Testovací metody, reprezentující jeden test zařazené v testovací třídě. Testy pak mohou assertovat výrazy pomocí standardních metod pro asserce ve třídě `Debug` ve jmenném prostoru `System.Diagnostics`. Příklad testu napsaného pro MS Test je v příloze 2. Test kontroluje, že objekt typu `FacebookPostInfo` má po vytvoření nulové ID.

Tento framework je velmi rigidní a neposkytuje příliš mnoho volnosti. Například nutnost specifikovat zdroje dat zásadně v externím XML souboru nepřispívá použitelnosti. Nicméně to, že je dodáván přímo s Visual Studiem a je tedy připraven k použití bez jakékoliv instalace či zásahu je nesmírnou výhodou.

4.2 Nunit

Nunit je knihovna pro tvorbu unit testů na platformě .Net. jedná se o port populární knihovny Junit pro Javu, nicméně Nunit prošel kompletním re-designem a využívá veškerých výhod platformy .Net jako například atributy v meta-datech.

Testy jsou strukturované podobně jako u MS Testu, nicméně poskytují větší flexibilitu. Každá testovací třída může mít definovánu metodu, která připraví prostředí pro všechny testy v dané třídě a ve třídách vnořených a také metodu, která připraví prostředí předtím, než je spuštěn každý jednotlivý test v dané třídě. Stejné možnosti jsou pak následně na konci každého testu či po skončení spouštění všech testů ve třídě. Takováto metoda má běžně za úkol uvést prostředí do původního stavu i za předpokladu, že test selhal. Velmi užitečnou novinkou je také třída označená atributem `SetUpFixture`, která nabízí možnost spustit určitý kód předtím, než je vykonán jakýkoliv test, a poté, co všechny testy doběhly. Tato flexibilita je zásadní pro užití knihovny pro správu automatizovaných testů UI.

Možnosti a flexibilita mohou být i na obtíž. Pokud by byla knihovna tak komplikovaná, že je těžké ji podporovat a v tomto konkrétním případě spouštět testy napsané za pomocí Nunity žádný, nebo téměř žádný nástroj na spouštění testů by ji nepodporoval. Naštěstí toto není případ Nunit. Existuje několik velmi povedených test-runnerů pro sady testů psaných za pomocí Nunit a dokonce vznikl i doplněk do Visual Studia, který umožňuje spouštět Nunití testy, jako by to byly testy pro MS Test. Síla komunity okolo Nunit je obrovské a vývojáři si rádi zpříjemní práci s něčím, co často používají, a proto je Nunit jasnou volbou pro správu naší sady.

5 Automatizujeme reálné uživatelské scénáře v Kentico CMS

Konkrétní návrh a implementaci frameworku a sady testů budeme demonstrovat na webové aplikaci Kentico CMS. Následující kapitola představí Kentico CMS a rozebere jeho architekturu, na základě které pak vytvoříme návrh architektury.

5.1 Kentico CMS

Kentico CMS je produkt zaměřený na středně velké a velké firmy. Zkratka CMS označuje v češtině správu obsahu, nicméně webová aplikace postavená na Kentico v základu poskytuje daleko více. Plně vybavený online-marketing modul a možnosti pro e-commerce (výstavbu e-shopů a podobně), vestavěné možnosti integrace s vybranými službami třetích stran, nebo třeba podpora funkcionality nutné pro výstavbu intranet-portálů, jako je například autentizace přes výzvu 401 jsou také součástí Kentico.

5.1.1 Architektura obecně

Kentico organizuje data do kategorií, podle typu objektu. Každý typ objektu má svůj ‘ObjectType‘, což je jedinečný textový identifikátor typu objektu o který se jedná. Jeden typ objektů je ukládán do jedné tabulky v databázi, ale je možné ukládat více typů objektů do jedné tabulky. Děje se tak však jen výjimečně, jsou-li například rozlišovány objekty pouze na základě toho, mají-li v určitém sloupečku hodnotu, či nikoliv. Typickým objektem v Kentico je například uživatel. Všichni uživatelé jsou v jedné tabulce, kde jsou jen a pouze uživatelé. Všechny objekty jsou v jedné databázi. Samotné typy objektů jsou také uloženy v databázi v jedné tabulce a typ objektů je sám o sobě typ objektů. Tabulka pro data o typech objektů, obsahuje důležité informace o jednotlivých typech objektů, jako například jejich atributy, a informace o nich. Díky tomu lze s každým objektem

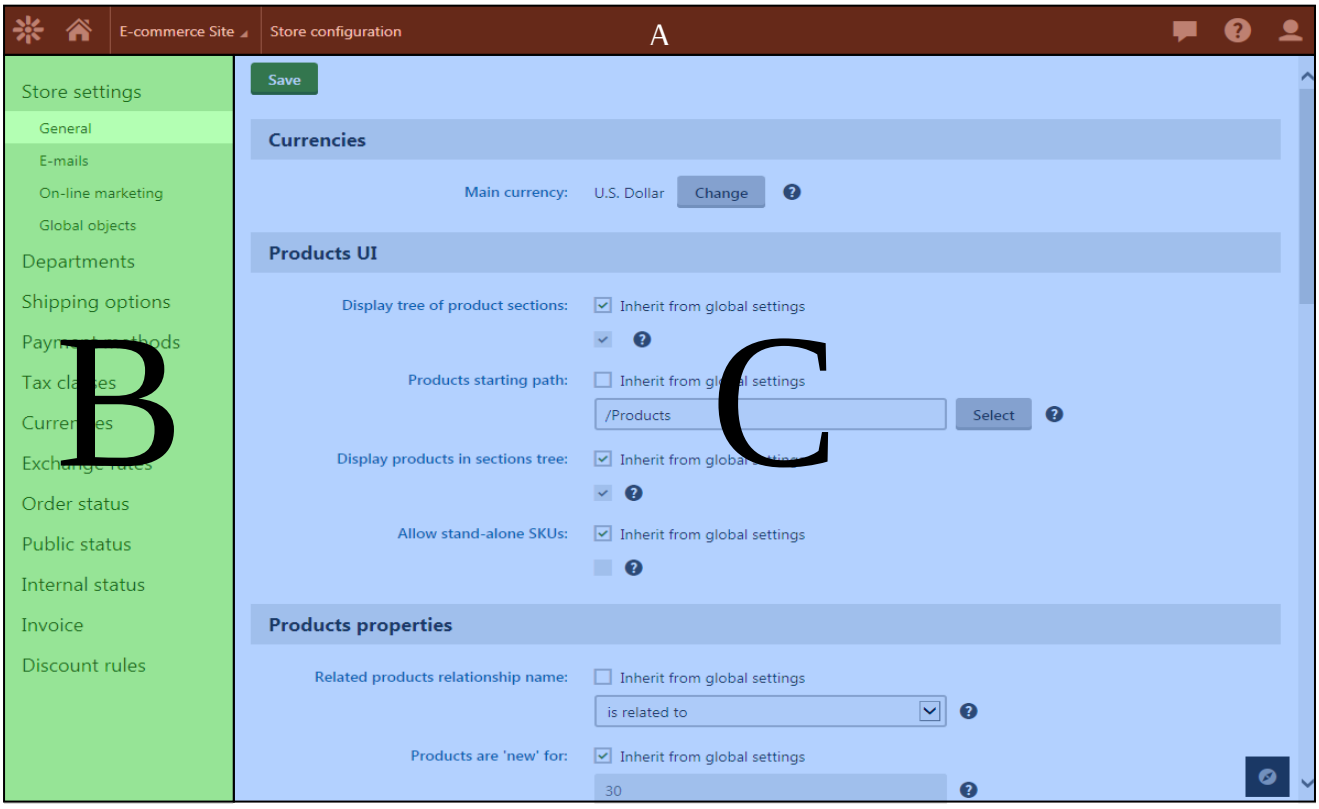
pracovat zcela obecně jen na základě jeho definice z databáze, což umožňuje obecnou architekturu UI.

5.1.2 Architektura UI

Kentico má již plnou podporu pro data-driven UI. V praxi to znamená, že informace o organizaci uživatelského rozhraní je uložená v databázi. Celý systém ještě nebyl převeden na tento model, ale z velké části tomu tak je a všechny nově vznikající části uživatelského rozhraní již vznikají jako data-driven UI.

5.1.2.1 UI element

Základní stavební jednotkou uživatelského rozhraní je UI element. UI element je zároveň také jeden ‘ObjectType‘. Všechny UI elementy jsou tedy v jedné tabulce. Samotné uživatelské rozhraní má však stromovou strukturu a to je potřeba zohlednit při ukládání UI elementů do databáze. Jeden UI element je pro uživatele reprezentován jednou stránkou administračního rozhraní v obsahovém rámci administrace. UI element je zkonstruován na základě šablony, která má definici taktéž v databázi, a jejích vlastností. Vše si ukážeme na příkladu – Obrázku 5.1.



5.5.1 - Ukázka administračního rozhraní Kentico 8

A. Hlavní navigační lišta:

Obsahuje informace o aktuálním umístění a možnost vrátit se do některé z nadřazených

položek. Také obsahuje důležité zkratky pro přístup ke kontextové nápovědě a na dashboard, který slouží k rychlému přístupu k vybraným aplikacím.

B. Navigační lišta:
   Obsahuje seznam UI elementů aktuální úrovně, popřípadě ještě o jednu úroveň níže jako tomu je v tomto případě u ‘General‘, ‘E-mail‘ a podobně. Slouží pro přepínání obsahu v obsahovém rámci C.

C. Rámec, obsahující samotný UI element. V tomto případě se jedná o šablonu pro formulář vygenerovaný na základě definice v databázi. UI elementy, které nejsou vygenerovány na základě šablony, mají také záznam v databázi. Zde je však uvedeno pouze umístění ASPX souboru, který daný element reprezentuje a plně jej definuje.

5.1.2.2 Ovládací prvek

UI element zdaleka není nejmenší stavební jednotkou uživatelského rozhraní. Šablony jsou sestavené z ovládacích prvků, které mají svůj základ také v databázi. Obecně se dá říci, že máme 2 hlavní typy ovládacích prvků:

1) Prvek je reprezentován ASCX souborem. Má svůj fyzický soubor a vlastní implementaci. Také má ale svoji definici v databázi, kde má definovanou množinu vlastností, které je možné danému prvku nastavit. Tyto vlastnosti jsou následně zohledněny při implementaci. ASCX-ové prvky jsou také často skládány do složitějších prvků, které plní konkrétní funkci, stejně tak jako v běžném přístupu při vývoji aplikace ve Web Forms.
2) Ovládací prvek nemá ASCX soubor. Takovýto prvek je v podstatě upřesněním nějakého obecnějšího prvku, který má svoji implementaci. Opakuje-li se příliš často určité použití obecného prvku, odštěpí se jeho nová varianta nakonfigurovaná přímo pro toto konkrétní použití. Typickým příkladem může být například ‘SiteSelector‘, který poskytuje výběr z dostupných portálů běžících v jedné instanci Kentico. Tento prvek je odštěpený z tzv. UniSelectoru, který umožňuje výběr jakéhokoli objektu. Typ vybíraných objektů, způsob výběru, návratová hodnota a další vlastnosti, které je nutno specifikovat, byly zafixovány u SiteSelectoru, protože je velmi často používán stále stejným způsobem.

5.1.2.3 Testovatelnost

Nehledě na to, že v minulosti se na testovatelnosti psaného kódu příliš nelpělo, většina nově vznikající funkcionality je psána s testovatelností jako hlavní prioritou. Momentální architektura však nahrává vkládání velkého množství logiky do prezentační vrstvy, tedy ovládacích prvků. Například právě UniSelector dovoluje spoustu nastavení a modifikací a tyto možnosti a všechny jejich možné kombinace je nutné zpracovat přímo v ovládacím prvku samotném, neboť ovlivňují právě jej. Když k tomu připočteme ještě různé kontextové informace, které prvek zohledňuje, je jasné, že implementace takového prvku je velmi složitá. Nezbývá tedy, než testovat prvek přímo tam, kde se

používá v reálných scénářích, poněvadž není možné a ani vhodné testovat všechny možné kombinace, jelikož většina z nich je nesmyslná a irelevantní.

5.2 Návrh a implementace

Architektura automatizačního frameworku do jisté míry kopíruje architekturu uživatelského rozhraní samotného produktu. Znamená to, že ke každému ovládacímu prvku, který budeme chtít ovládat, vytvoříme řídící objekt a řídící objekty budou mezi sebou mít do značné míry stejné vztahy, jako jejich přidružené ovládací prvky. Samotné řídící objekty však žádné testy nevykonávají. Slouží pouze k řízení ovládacích prvků. Je potřeba také platforma pro testy samotné. Tu nám zajišťuje Nunit, ale jak bylo již popsáno výše, Nunit je nástroj pro tvorbu unit-testů a automatické testy UI vyžadují odlišný přístup. Vznikla proto ještě robustnější platforma uzpůsobená přímo pro Testy UI.

Celkově je řešení rozděleno na několik projektů. Jádro neboli platforma, kde je obecná a podpůrná funkcionalita. K němu je přidružen projekt testů platformy, který obsahuje sadu testů, které mají zajistit, že platforma funguje správně. Pro každou verzi produktu Kentico pak existuje jeden projekt, který obsahuje sadu scénářových testů pro tuto verzi a množinu řídících tříd. Na odevzdaném CD v příloze 3 je k dispozici pouze projekt přidružený k verzi 8, neboť také pouze tato verze produktu je nyní k dispozici zdarma a je nainstalována na virtuální applianci.

Následující podkapitoly podrobněji rozebírají architekturu sady řídících tříd a platformy pro testy samotné.

5.2.1 Testovací platforma

Již několikrát bylo zmíněno, že pro spouštění jednotlivých testů využíváme Nunit. Pro potřeby UI testů bylo však nutné z této platformy využít, co se dalo, abychom dosáhli kýžené robustní platformy a tvořit jednotlivé testy tak bylo co nejjednodušší. Aby mohl autor testu okamžitě začít ve své testovací metodě popisovat testovaný scénář, je nutné inicializovat celé prostředí. Test musí vědět, jaká instance webové aplikace se testuje, kde se nachází, je potřeba inicializovat log, instanci vybraného internetového prohlížeče a také je potřeba zpracovat další volitelná nastavení celého testovacího sezení. Vše pak souvisí s tím, jak Nunit volá metody označené atributy. Proto následující podkapitola podrobněji představí jednotlivé klíčové atributy.

5.2.1.1 Klíčové atributy Nunit

`TestFixture` – Tento atribut označuje třídu, která obsahuje jednotlivé testy a také volitelnou připravovací či ukončovací metodu. Dále budeme takové třídy nazývat testovacími třídami. Příklad použití tohoto atributu je ukázka 5.2.

`Test` – Atribut Test označuje přímo testovací metodu. Tedy metodu, která popisuje testovaný scénář. Ukázka 5.2 ilustruje jeho použití.

```
[TestFixture]
public class SuccessTests
{
  [Test]
  public void Add()
  { /* ... */ }
}
```

5.2 - Ukázka použití atributů TestFixture a Test

`TestFixtureSetUp` – Třída označená atributem `TestFixture` může mít definovánu metodu označenou atributem `TestFixtureSetUp`. Metoda takto označená je zavolána jednou, předtím, než začne jakýkoliv test zařazený to této testovací třídy. Dědí-li testovací třída z jiné testovací třídy, jsou metody označené `TestFixtureSetUp` spuštěny hierarchicky. Tedy dědí-li A od B, první se spustí metoda třídy B a potom metoda třídy A.

`TestFixtureTearDown` – `TestFixtureTearDown` má přesně opačný účel než předchozí atribut. Zatímco 'SetUp' má připravit prostředí pro testy, které budou následovat, 'TearDown' má zajistit, že prostředí zůstane v původním stavu, aby tím nebyly ovlivněny jiné testy. Při dědění se metody spouští přesně v opačném pořadí než 'SetUp' metody.

`SetUp` – Stejně jako testovací třídy mají metodu pro inicializaci i jednotlivým testům lze připravit prostředí. A pak jej zase uvést do původního stavu. `SetUp` je označení pro metodu definovanou uvnitř testovací třídy. Metoda takto označená je spuštěná před každým jednotlivým testem v dané testovací třídě, nebo třídách, které z ní dědí. Pořadí spouštění 'SetUp' metod je analogické k `TestFixtureSetUp`.

`TearDown` – Analogicky k testovacím třídám, `TearDown` je označení pro metodu, která uklízí prostředí po každém testu. Měla by být navržena tak, aby dostatečně zametla stopy po každém testu, který obsluhuje i v případě neúspěchu. Jinak by následující testy mohly selhávat právě kvůli nečekanému stavu prostředí.

`SetUpFixture` – `SetUpFixture` je označení pro třídu, která může obsahovat metody s označením `SetUp` a `TearDown`. Testovací metody ani jiné nesmí takto označená třída obsahovat. Metody slouží k inicializaci a úklidu všech testů, či testovacích tříd ve stejném jmenném prostoru. Rozdílem oproti dědění je ten, že při použití `SetUpFixture` se inicializace vykoná pouze jednou pro všechny testovací třídy ve jmenném prostoru, zatímco dědí-li více testovacích tříd od jedné, provede se pro každou jednotlivou třídu inicializace i úklid. Není-li třída označená `SetUpFixture` zahrnuta v žádném jmenném prostoru, platí její metody pro celý projekt. To je nejširší oblast, pro kterou může `SetUpFixture` mít efekt.

5.2.1.2 Využití atributů ve frameworku

Pro celkovou inicializaci testovacího sezení je použita třída s označením `SetUpFixture` bez jmenného prostoru. Inicializační metoda zajišťuje vytvoření adresáře pro výstup logu a základní inicializaci kontextu. Úklidová metoda uzavírá a ukončuje logování. Celkové řešení je rozděleno do několika projektů a proto je potřeba mít v každém z nich tuto inicializační třídu. Jelikož chceme použít stejnou společnou inicializaci v několika projektech, máme v každém projektu prázdnou třídu s atributem `SetUpFixture`, která dědí ze společné inicializační třídy, která obsahuje implementované inicializační a úklidové metody. Takto se inicializační kód nachází pouze na jednom místě a nastane-li situace, že je potřeba jej změnit, stačí tak učinit pouze na jednom místě. Také je díky tomu možné inicializaci či úklid rozšířit pro potřeby jednotlivých projektů se zachováním obecnosti pro všechny jmenné prostory.

Všechny testovací třídy dědí ze třídy `BaseTest`. Třída `BaseTest` zajišťuje rozšiřujícím třídám pohodlný přístup ke klíčovým datům z nastavení, či kontextu a zároveň zajišťuje správné načtení nastavení a inicializaci kontextu pro jednotlivé třídy. To vše díky metodám označeným `TestFixtureSetUp` a `TestFixtureTearDown`, které jsou spuštěny vždy před každou třídou testů. Vše je velmi důležité, protože na tyto akce spoléhá speciálně přizpůsobený systém logování průběhu a výsledků testů a také je tímto umožněno jednoduše měnit globální nastavení testů na jednom místě v čitelném a zdokumentovaném XML souboru. Metody pro inicializaci a úklid jednotlivých testů také zajišťují správné chování losovacího systému, ale také pomáhají přepínat kontext z jednoho testu na druhý. Díky tomu lze každému testu dát individuální nastavení kterékoliv položky z globálního nastavení. Dovoluje to tak autorům testu specifikovat nutné nastavení, se kterým test počítá ať už dočasně pro potřeby vývoje daného testu, či nikoliv. Příkladem může být test, který kontroluje správné chování řídícího prvku pro vysouvací menu. Test se jmenuje `DropDownTest` a lze jej nalézt na přiloženém CD – příloze 1. Tento test spoléhá na to, že se nachází na speciální testovací stránce. Proto je test nastaven tak, aby probíhal vždy jen na této testovací stránce a ignoroval globální nastavení pro vykonávání klasických scénářových testů.

5.2.1.3 Další podpůrné mechanismy

Selenium nabízí v základu řízení pro Firefox a v této práci byla přidána ještě podpora pro Internet Explorer a Chrome. Prohlížeče řídí třídy, které implementují rozhraní `IWebDriver` a jsou součástí knihovny Selenium. V naší implementaci je řídící zastoupen třídou `WebDriver`, která zaštiťuje veškeré nutné operace rozhraní `IWebDriver`. Účelem bylo cíleně zastřít přímý přístup k řídícímu objektu Selenia hned z několika důvodů.

1) Větší kontrola nad jednotlivými akcemi. Je tak možné zcela automaticky logovat důležité akce, které test vykonává, aniž by na to musel autor testu myslet. Také to do značné míry zkracuje kód testu, čímž jej činí stručnější a čitelnější.

2) Kompatibilita s vlastními řídícími objekty. Řídící objekty našeho frameworku nejsou kompatibilní s těmi ze Selenia. Aby s nimi řídící objekt mohl pracovat, je potřeba mezivrstva, která předá již kompatibilní objekt řídícímu objektu Selenia.
3) Rozšiřující metody. Do `WebDriver`u byly doimplementovány rozšiřující metody pro usnadnění práce s ním. Jako příklad uvedeme metodu `SwitchToParentFrame`, která slouží pro přepnutí kontextu do rodičovského rámce na stránce a není obsažena v rozhraní `IWebDriver`.

Již několikrát byl zmíněný speciálně upravený logovací systém. Autorům samotného testu nabízí prostředky pro zaznamenávání informativních, výstražných či chybových hlášek. Každá hláška pak může být doprovázena snímkem celé stránky. To je obzvláště vhodné u chybových hlášek. Jeden snímek stránky s vhodnou hláškou mohou objasnit, proč test selhal. Pořizování snímků je nastavitelné jak v globálním nastavení, tak pro každý jednotlivý test na tři úrovně: zapnuto, vypnuto a jen chyby. Při zapnutém snímkování se snímek vytvoří při vytváření jakékoliv hlášky, při vypnutém se naopak netvoří žádné. Je-li zapnuto snímkování jen pro chyby, výchozí chování je takové, že se snímky tvoří jen se záznamem chybové hlášky. Při volání logovací metody je možné nastavení přebít volitelným argumentem, který určuje, zda pořídit snímek. Vše pak ještě doplňuje možnost smazat snímky obrazovky pro úspěšné testy. Tím se zajistí, že při velké sadě testů nebude výstup losovacího systému zbytečně velký. Na závěr je nutné zmínit, že výstup z logu není obyčejný textový soubor, neboť obrázky by do něj nebylo možné zakomponovat. Proto byl pro výstup zvolen formát HTML. Snímky obrazovky jsou tak zobrazeny přímo u hlášek, ke kterým patří a celý výstup je barevně rozlišen.

Každý unit test obsahuje nějakou aserci. Testy uživatelského rozhraní obsahují asercí daleko více. Během vykonávání jednoho scénáře je běžné kontrolovat několik věcí. Často se také stane, že chceme ověřit několik věcí zároveň ale přitom každou zvlášť, abychom měli možnost zaznamenat podrobněji chybové hlášky. Místo chyby, která říká, že jedna z určité množiny věcí je špatně chceme vědět, které věci jsou špatně a kolik jich je. K tomu je běžný systém asercí nedostačující. Chceme-li například zkontrolovat, že formulář obsahuje určitá pole, potřebujeme zkontrolovat všechna pole, ačkoliv chybí již první z nich. Vznikla proto statická třída `AssertAll`, která obsahuje jedinou metodu, která vykoná všechny asserce předané parametrem a je-li alespoň jedna neúspěšná, vyhodí sama `AssertionFailed` výjimku a jako zprávu jí předá veškeré chybové hlášky všech selhaných asercí.

Pro kompletní možnosti ovládání obsahuje Selenium třídu `Actions`. Vzhledem k nekompatibilitě řídících objektů vznikl v našem projektu stejnojmenný adaptér. `Actions` zajišťují operace, jako je drag & drop, dvojklik nebo ovládání klávesnicí.

5.2.2 Řídící prvky

Jednotlivé ovládací prvky jsou řízené řídícími objekty. Stejně tak jako složité ovládací prvky jsou skládány z jednodušších prvků, jsou řídící objekty složitých prvků skládány z instancí jednodušších řídících tříd. To nám dovoluje vyhnout se opakování kódu a zároveň tak zajistit, že změní-li se jeden ovládací prvek, jednoduše se mu přizpůsobí řídící třída a není potřeba změnu reflektovat na více místech.

Selenium pro obecnou reprezentaci jednoho řídícího objektu používá rozhraní `IWebElement`. Bylo by vhodné jej také použít v zájmu kompatibility s prvky selenia, ale rozhodli jsme se `IWebElement` raději zahrnout v našem obecném řídícím objektu jako soukromý člen `mElement` a zpřístupnit jej navenek veřejným členem `InternalElement`. Implementovat celé rozhraní `IWebElement` nebylo potřeba a jevilo se náročné. Tím, že je `IWebElement` však stále dostupný jako člen, je možné jej kdykoliv z našeho řídícího prvku vytáhnout a poslat jako argument metodám ze Selenia. Jelikož však takovéto použití není zamýšlené a tato možnost existuje pouze jako poslední záchrana, jsou naše řídící prvky vybaveny vším potřebným z rozhraní `IWebElement`. Abychom toto zajistili, veškeré řídící prvky dědí ze třídy `AbstractElement`. Rozhodnutí neimplementovat `IWebElement` nakonec přispívá i jednoznačnosti identifikace volaných metod, neboť naše metody si neporadí s `IWebElement` a metody Selenia zase neumí pracovat s `AbstractElement`.

Třída `AbstractElement` je abstraktní. To znamená, že nelze vytvořit čistě její instanci. Lze vytvořit pouze objekty tříd, které z této třídy dědí a implementují virtuální členy. Bohužel C# neumožňuje definovat virtuální konstruktor a proto není možné zajistit, aby všechny řídící třídy měly základní bez-parametrický konstruktor. Při použití v generických metodách lze však vyžadovat konkrétní typ a dokonce i konstruktor, jak ilustruje ukázka 5.3. Metoda slouží k vyhledání prvku na

```
public T FindElement<T>(By by = null) where T : AbstractElement,
new()
```

5.3 - Ukázka syntaxe požadavků u generické metody

stránce a vrací k němu přidružený řídící prvek. K tomu aby prvek byl vrácen, musí projít validací, která se dělá pomocí instanční metody. K tomu je potřeba vytvořit prázdnou instanci typu `T`, a proto je vyžadován základní konstruktor. Jazyk C# poskytuje prostředky pro to, aby veškeré metody pracující s řídícími objekty měly za každých okolností veškeré dostupné členy.

Zcela základní a obecnou řídící třídou, jejíž instanci lze vytvořit je `BaseElement`. `BaseElement` dovoluje Jakýkoliv úkon s jakýmkoliv ovládacím prvkem. Dokáže ovládat cokoliv od obyčejného textového pole či obyčejného odkazu až po nejsložitější skládané ovládací prvky. Jakkoliv je ale obecný, práce s ním není příliš pohodlná. Jako příklad mezi prací s `BaseElement` a `Table`, který slouží pro obsluhu tabulek, poslouží ukázka 5.4. První řádek ukazuje práci s obecným

```
string text = Driver.FindElement(By.TagName("Table"))
      .FindElement(By.XPath("./tr[3]"))
      .FindElements(By.TagName("td"))[2].Text;

string text = Driver.FindElement<Table>()
      .Rows[2].Cells[2].Text;
```

5.4 - Ukázka použití obecného a silně typovaného prvku

prvkem, zatímco druhý využívá výhod silně typovaného řídícího prvku pro tabulky. Oba řádky přitom přiřazují do proměnné `text` obsah třetí buňky ve třetí řadě první tabulky na stránce. Cílem automatizačního frameworku je, aby jeho uživatel, tedy autor finálního testu nemusel nikdy sáhnout po třídě `By`. Tedy aby nemusel manuálně vyhledávat jednotlivé prvky. Místo toho mu bude vše zpřístupněno pohodlně pomocí členů nadřazených řídících prvků a bude-li chtít vyhledat konkrétní ovládací prvek na stránce, použije silně typovanou metodu `FindElement<T>`.

Tvorba nové řídící třídy probíhá tak, že její autor vytvoří novou třídu, která rozšiřuje `BaseElement` a naimplementuje nezbytně nutné členy:

`Validate()` – Metoda, která validuje, že přidružený prvek DOM stromu je vskutku možné řídit tímto objektem.

`DEFAULTXPATH` – Člen, který udává obecné pravidlo formátu XPath pro řízený prvek.

Je-li to nutné, upraví autor třídy konstruktory zděděné od `BaseElement`. Následně je možné třídu rozšířit o specializované úkony, či přepsat úkony základní.

5.2.3 Napojení na API

Ačkoliv píšeme sadu testů uživatelského rozhraní, je napojení na API testovaného produktu velkou výhodou. Existuje-li tato možnost, je vhodné ji využít. Naše sada testů je na API napojena přímo referencemi knihoven produktu. Není tak potřeba používat vestavěné REST-API či jiných složitějších služeb. Díky tomu je možné některé operace, jako třeba naplnění databáze testovacími daty vykonat přes API, tedy daleko rychleji, než automatizovaným vyplňováním a odesíláním formulářů. Samozřejmě že je-li předmětem testu otestovat scénář tvorby uživatele, nebudeme volat API pro jeho vytvoření, ale celý scénář bude vykonán z pohledu uživatele. Nicméně chceme-li otestovat zobrazení seznamu uživatelů, filtrování seznamu atd. při velkém počtu položek, naplníme databázi uživateli přes API a test tak poběží výrazně kratší dobu. Takové testy však není možné vykonávat v Selenium Gridu, pokud ovšem na výkonných strojích není prázdná instance Kentico se správně nakonfigurovaným připojením k databázovému serveru.

5.3 Konkrétní příklad

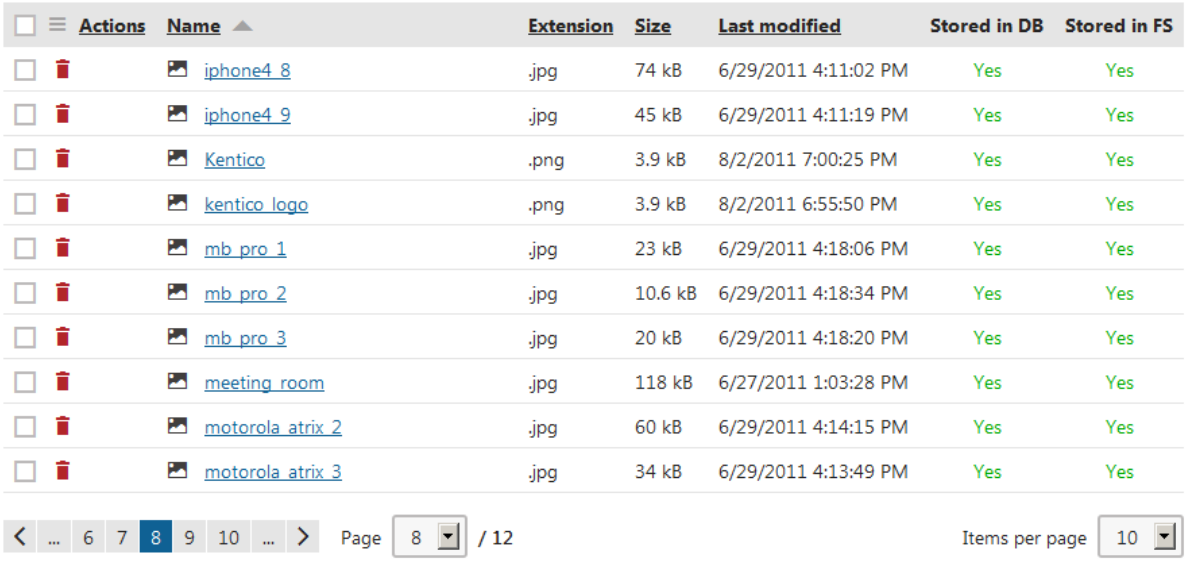
Na dvou příkladech si nyní ukážeme, jak vytvořit řídící třídu pro komplikovaný ovládací prvek tak, aby byla lehce udržovatelná a bylo možné ji rychle přizpůsobovat a na druhém si ukážeme případ, kdy již hotový test jsme kompletně upravovali z verze 7 na verzi 8.

5.3.1 UniGrid

Jako první si na reálném příkladu ukážeme přístup při implementaci řídícího prvku pro ovládací prvek zvaný UniGrid.

5.3.1.1 Popis UniGridu

UniGrid je ovládací prvek, který slouží pro zobrazení seznamů prvků. Podporuje velké množství variací a nastavení. Následuje obrázek 5.5, který vyobrazuje UniGrid v jedné z konfigurací a který popíšeme a vysvětlíme, co jednotlivé prvky ovládací prvky dělají.



5.5 - Snímek UniGridu

Snímek ukazuje UniGrid použitý pro zobrazení seznamu příloh k dokumentům v systému. Hlavička tabulky obsahuje zcela přirozeně názvy sloupečků. Kliknutím na podtržený název sloupečku lze seznam řadit podle hodnot v daném sloupečku. Výjimkou je sloupeček ‘Actions‘, který neobsahuje data a nejde tedy podle něj řadit. Tento sloupeček je společný pro všechny UniGridy a má všude stejný název. Namísto dat se v něm zobrazují ikonky akcí, které lze s každým prvkem provést. Zde vidíme ikonku v podobě odpadkového koše, která slouží pro mazání položek. Mezi další obvyklé akce patří editace či detailní náhled. Ještě více vpravo od sloupečku s akcemi je sloupeček pro násobnou selekci. Tento sloupeček není povinný ve všech seznamech. Slouží pro vybírání položek pro násobné akce, které se spouští pomocí jiného ovládacího prvku. Sloupečky s daty jsou v UniGridu zvláštní tím, že dovolují jednoduché automatické transformace, jako je například odkaz na soubor ve

sloupečku ‘Name‘, nebo lokalizace booleovských hodnot ve sloupečcích ‘Stored in DB‘ a ‘Stored in FS‘.

Naspodu UniGridu jsou ovládací prvky pro stránkování. Čtverečková políčka vlevo slouží pro listování. Vedle se nachází ovládací prvek pro výběr konkrétní stránky. Ten není povinný a jeho podoba závisí na počtu stránek. Pro malý počet stránek není zobrazen vůbec. Pro střední počet je zobrazen ve formě vysouvacího menu a pro velký počet stránek je zobrazen ve formě obyčejného textového pole. Vpravo pak vždy nalezneme vysouvací menu pro výběr počtu prvků na stránku.

5.3.1.2 Popis implementace řídící třídy UniGridu

Abychom mohli přejít k popisu problémů, které zapouzdřením řízení celého UniGridu řešíme za autora testu, popíšeme stručně implementaci řídící třídy.

Řídící třída pro UniGrid se nazývá `UniGridGrid`. Hlavní částí gridu je tabulka s obsahem a autorovi testu musíme poskytnout jednoduchý a transparentní přístup k datům v tabulce. Člen `Rows` tedy zpřístupňuje kolekci prvků typu `UniRow`, tedy prvků pro řízení jednotlivé řady. Kolekce není udržována v paměti řídícího prvku pro UniGrid (není ‘kešovaná‘) a je-li potřeba více přístupů po sobě k této kolekci, je vhodné optimalizovat tak, že si kolekci autor testu uloží do lokální proměnné. Obecně není uchovávání referencí na prvky vhodné. Problémy s tím spojené budou diskutovány později.

Výjimkou z kolekce řádků je hlavička tabulky. Ta je zpřístupněná zvlášť členem `HeadRow` a poskytuje přímý přístup k hlavičce tabulky. Ke kolekci prvků pro řízení buněk hlavičky lze přistoupit skrz člen `Cells`, nebo lze indexovat přímo `HeadRow`. Jednotlivé prvky v `Cells` jsou typu `UniHeadCell` a dovolují řídit speciální funkcionalitu hlavičkových buněk. Pomocí členu `SortedBy` lze zjistit, jestli je tabulky řazena podle tohoto sloupce a jakým směrem. `CanSort` zase vrací informaci o tom, zda je možné podle tohoto sloupce vůbec řadit. Také byla speciálně upravena vlastnost `Text`, aby skutečně vždy vracela název sloupce.

Kolekce jednotlivých řádků je tedy již lehce přístupná. Jednotlivé řádky je však také potřeba vybavit prostředky pro snadnou interakci. Hlavní je kolekce prvků pro řízení jednotlivých buněk nazvaná `Cells`, která poskytuje obdobně jako `Rows` přístup ke kolekci buněk. Objekt typu `UniRow` je také možné indexovat přímo a to jak celým číslem, tak názvem sloupce. Autor testu tak jednoduše získá hodnotu sloupečku v libovolné pouze na základě jeho názvu.

`UniGridGrid` také zpřístupňuje řídící prvek pro stránkování skrz veřejný člen `Pager`. Objekt typu `Pager` umožňuje zjistit nebo změnit počet prvků na stránku pomocí vlastnosti `SelectedPaging`. K tomu je potřeba si vybrat jednu z dostupných možností, které zpřístupňuje člen `PagingOptions`. Aktuální stránku lze zjistit nebo změnit pomocí členu `ActivePage`.

Přímo z vrchní úrovně řídícího prvku pro UniGrid jsou dále dostupné základní informace o tabulce jako je seznam názvů sloupců a poskytuje autorovi testu jednoduše seřadit seznam podle

libovolného sloupce pomocí metody `SortBy()`, která si jako parametr žádá název sloupce a směr řazení a vrací booleovskou informaci o tom, jestli uspěla. Je pak na autorovi testu, aby zpracoval výsledek. Měl-li být pokus o seřazení neúspěšný, je negativní výsledek žádaný.

5.3.1.3 Problémy

Díky tomu, že autorovi konečného testu poskytujeme již ucelené řešení pro řízení UniGridu, zbavujeme ho zodpovědnosti za velké množství logiky, usnadňujeme mu do značné míry psaní testu a hlavně se vyvarujeme opakování kódu a zlepšujeme udržovatelnost sady testů. Řídící třída `UniGridGrid` také řeší za pisatele testů spoustu problémů. UniGrid je ajaxový ovládací prvek. To znamená, že aniž by vynucoval načítání celé stránky znovu, dynamicky mění svůj obsah. To přináší nemalé problémy při řízení takového prvku. Vynutí-li si nějaký úkon znovunačtení celé stránky, Selenium zajistí, že následující úkon počká na dokončení načítání. S asynchronní změnou obsahu ale přichází problém. Samotný driver prohlížeče totiž nečeká na dokončení operace a tak je třeba manuálně čekat, než se změna obsahu provede. To lze detekovat, je však potřeba zvolit vhodnou podmínku. Další problém s dynamickými změnami obsahu stránky je, že prvky ze stránky mizí a objevují se jiné. S tím je spojená výjimka `StaleElementReferenceException`, kterou Selenium vyhodí, projeví-li test snahu o práci s prvkem, který již není na stránce. V případě UniGridu se tento problém projevil již při pokusu o řazení položek v tabulce. Když je proveden klik na název sloupečku, dle kterého lze řadit, celý UniGrid se překreslí. To znamená, že `mElement` `UniGridGrid`u po jakémkoliv pokusu o řazení neplatný a jelikož na práci s ním závisí jakákoliv práce s UniGridem, jeví se situace jako bezvýchodná. Pro pisatele testu by tak bylo nutné po prvním kliknutí na název sloupce zahodit celou instanci řídícího prvku a pořídit nový. Samozřejmě až poté, co zajistí, že počká na dokončení asynchronní operace. Zcela stejný problém nastává při změně stránky, nebo počtu prvků na stránku. To zdaleka není pohodlné ovládání pro uživatele frameworku. Díky tomu, že si jakkoliv řídící prvek, který dědí z `BaseElement` pamatuje, jak byl vyhledán, je možné jej pomocí stejného vyhledávacího výrazu vyhledat znovu. A díky tomu, že Seleniovský řídící prvek `IWebElement` je pouze vnitřní člen, je možné jej jednoduše nahradit. Samozřejmě, že se tak děje poté, co je zajištěné, že skončilo dynamické načítání obsahu. Pro pisatele testu je tak vše dokonale zapouzdřeno. Pomocí jedné a té samé instance řídící třídy `UniGridGrid` může ovládat UniGrid bez rizika vypršení platnosti prvku.

Jako ukázka interakce s třídou `UniGridGrid` byly sepsány testy ve třídách `DocumentTypeTests` a `EventLogTests`. Tyto testy testují seznam typů dokumentů a seznam událostí v systému a kontrolují vlastnosti těchto seznamů.

5.3.2 ContentTreeTest

`ContentTreeTest`, je testovací třída, která existuje pro verzi 7 i 8 Kentico CMS a obsahuje jeden test. Na popisu jeho transformace z jedné verze na druhou si ukážeme stabilitu a udržovatelnost našeho řešení. Obě verze tohoto testu lze najít na přiloženém CD v příloze 1.

Test, který třída obsahuje je scénář, ve kterém uživatel vytvoří dokument typu ‘Folder‘ a následně jej opět smaže. Test také kontroluje, že uživatelské rozhraní se chová správně během vykonávání tohoto scénáře.

Samotná změna z verze 7 na verzi 8 nebyl triviální, neboť mezi před vydáním verze 8 prošel celý systém kompletním re-designem a uživatelské rozhraní tak dostalo zcela novou tvář. Bylo tak nutné upravit nejen `DEFAULTXPATH` a `Validate` metodu použitých řídících tříd, ale i jejich speciální funkcionalitu. Například třídě `ContentTree`, která ovládá strom dokumentů, bylo nutné upravit metodu `ExpandAll`, na rozvinutí celého stromu, neboť způsob, jakým se uživatelské rozhraní chová při rozvíjení jednotlivých uzlů, se změnil. Testovací třída ovšem zaznamenala jen kosmetické změny. Změnil se jmenný prostor, řádek pro navigaci do správy obsahu a řádek na potvrzení dialogu o smazání dokumentu, pro který ještě není systémové řešení, které by splňovalo podmínky návrhu, jelikož se jedná pouze o proof-of-concept verzi řešení.

Dá se tedy říci, že navzdory kompletní změně uživatelského rozhraní, test, který pracuje pouze se systematicky řízenými ovládacími prvky, zůstane stejný, nezmění-li se zásadním způsobem způsob vykonávání scénáře. Používá-li více testů tentýž řídící objekt, je potřeba jej upravit pouze na jednom místě, což výrazně přispívá udržovatelnosti testů. Obvyklé scénáře, jako je například přihlášení se do systému, nebo odhlášení jsou pokryty také předpřipravenými funkcemi. Nestane se tak, že by bylo potřeba upravit chování testu na více než deseti místech. Jakmile se vykonává stejná věc na více než pěti místech, jedná se o horkého kandidáta na vytvoření veřejně dostupné metody pro vykonání takovéto akce. Nastane-li například situace, že vytváření dokumentů bude velmi častou činností testů, vznikne metoda pro vytvoření dokumentu, která si bude žádat pouze název typu dokumentu a data nutná pro vyplnění formuláře. Změní-li se následně proces vytváření dokumentu, bude stačit upravit tuto metodu.

6 Závěr

Vývoj celého automatizačního frameworku pro tak rozsáhlý produkt jako je Kentico CMS je běh na dlouhou trať a projekt se nyní nachází na samém začátku. Ačkoliv se framework stále vyvíjí a přibývají nové řídící prvky a testů není zrovna mnoho, zkušenosti nasbírané během přepisování testů ze staré verze na novou ukazují, že architektura tak, jak byla navržená, je velmi dobrá. Testy bohužel vznikaly i přesto, že pro některé ovládací prvky ještě neexistoval řídící prvek a místo toho, aby autor

testu řídící prvek vytvořil, porušil filozofii návrhu. Dělo se tak zejména z důvodu úspory času a zjednodušení implementace. Kvůli tomuto bylo během přepisování testů potřeba upravovat i testy samotné. To však jen utvrzuje a zdůrazňuje správnost návrhu. Do částí testů, které respektovaly návrh architektury, nebylo potřeba zcela zasáhnout a stačilo pouze upravit jednotlivé řídící prvky tak, aby si poradili s novým uživatelským rozhraním.

Zkušenosti s vykonáváním testů a jejich investigací ukazují na ohromnou důležitost logování. Log musí být dostatečně obsáhlý, ale přehledný. Snímky obrazovky jsou u testování uživatelského rozhraní nutností. Posloupnost snímků obrazovky spolu s informacemi o vykonávaných krocích může při selhání testu poskytnout dostatek informací o tom, kde se stala chyba a ještě posloužit jako podklad pro nahlášení chyby do systému pro evidenci chyb, což ušetří mnoho času při jinak nutném krokování testu a hledání chyby. Zdokonalování logování bylo proto věnováno nemálo úsilí a je pravděpodobné, že z dalších zkušeností s prací s výstupem logování přibudou další vylepšení.

Z pohledu dalšího vývoje znamená rozšiřování frameworku nyní hlavně tvorbu řídících tříd a testů. Samotná platforma pro psaní testů je velmi robustní a poskytuje dostatečné zázemí. Ovládací prvky se budou automatizovat v pořadí podle toho, jak často se v systému vyskytují. Po UniGridu tak bude následovat UniSelektor a další generické ovládací prvky. Zároveň je také nutné rozšiřovat sadu testů, poněvadž samotné řídící třídy chyby v produktu neodhalí. Toto jsou dvě hlavní úrovně, na kterých se bude dále automatizace uživatelského rozhraní ve firmě Kentico rozvíjet.

Pro oblast mimo Kentico znamená tato práce hlavně úspěšný proof-of-concept projekt na automatizaci testování uživatelského rozhraní webových aplikací s popisem filozofie návrhu, který je aplikovatelný na jakýkoliv projekt, jehož uživatelské rozhraní je hierarchické podobně jako to v Kentico CMS.

7 Literatura

[1] MCWHERTER, Jeff. Testing asp.net web applications. 1st ed. Indianapolis, IN: WIley Pub., Inc, 2009, p. cm. ISBN 04-704-9664-9.

[2] OSHEROVE, Roy. The art of unit testing: with examples in .NET. Greenwich, CT.: Manning, 2009, xxiii, 296 p. ISBN 19-339-8827-4.

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

[4] Selenium History. [online]. [cit. 2014-05-15]. Dostupné z: http://docs.seleniumhq.org/about/history.jsp

[5] http://watin.org/

[6] http://htmlunit.sourceforge.net/

[7] http://code.google.com/p/selenium/wiki/Grid2

[8] http://msdn.microsoft.com/cs-cz/library/ms173156.aspx

[9] http://msdn.microsoft.com/cs-cz/library/hk5f40ct(v=vs.90).aspx

[10] http://msdn.microsoft.com/cs-cz/library/k535acbf(v=vs.71).aspx

[11] http://www.scrumcrazy.com/User+Story+Basics+-+What+is+a+User+Story%3F

[12] http://www.webdevelopersnotes.com/design/browsers_list.php3

[13] http://www.w3schools.com/browsers/browsers_stats.asp

//Na ukázku jsem vygeneroval citaci pro Selenium.org

8 Seznam příloh