
**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**
**BRNO UNIVERSITY OF TECHNOLOGY**

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**
**FACULTY OF INFORMATION TECHNOLOGY**

**ÚSTAV INFORMAČNÍCH SYSTÉMŮ**
**DEPARTMENT OF INFORMATION SYSTEMS**


# AUTOMATIZACE V PROJEKTECH VÝVOJE SOFTWAROVÝCH APLIKACÍ

**AUTOMATION IN SOFTWARE APPLICATION DEVELOPMENT PROJECTS**

**BAKALÁŘSKÁ PRÁCE**
**BACHELOR'S THESIS**


**AUTOR PRÁCE** **DANIEL MERTA**
**AUTHOR**

**VEDOUCÍ PRÁCE** **doc. RNDr. JITKA KRESLÍKOVÁ, CSc.**
**SUPERVISOR**



**BRNO 2017**

*Zadání bakalářské práce/19320/2016/xmerta00*

**Vysoké učení technické v Brně - Fakulta informačních technologií**

Ústav informačních systémů Akademický rok 2016/2017

# Zadání bakalářské práce

Řešitel: **Merta Daniel**
Obor: Informační technologie
Téma: **Automatizace v projektech vývoje softwarových aplikací**
**Automation in Software Application Development Projects**
Kategorie: Softwarové inženýrství

Pokyny:

1. Seznamte se s principy a možnostmi Definition of Done (DoD), z čeho vychází a k jakému účelu se používá.
2. Seznamte se detailně, jakým způsobem je DoD použito ve vývojovém prostředí projektů společnosti Kentico software, s.r.o. a co je jeho obsahem.
3. Analyzujte, které části DoD by bylo možné v prostředí společnosti Kentico automatizovat.
4. Po dohodě s konzultantem společnosti Kentico zvolte dílčí části DoD a navrhněte jejich automatizaci.
5. Implementujte navrženou automatizaci vybraných částí a použitelnost demonstrujte na vhodném vzorku dat vybraném po dohodě s vedoucí.
6. Zhodnoťte dosažené výsledky a diskutujte možnosti použití zvoleného postupu i pro další automatizaci v daném vývojovém prostředí.

Literatura:

- CRISPIN Lisa, GREGORY Janet. *Agile Testing*: Addison-Wesley, 2009. ISBN 978-0-321-53446-0.
- ALAM Afshar, PADENGA Tendai. *Application Software Reengineering:* New Delhi Pearson Education, 2010. ISBN 978-81-317-3185-7.
- Definition of Done [on line, cit. 2016-07-16].

Pro udělení zápočtu za první semestr je požadováno:

- Splnění bodů 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Kreslíková Jitka, doc. RNDr., CSc.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
*vedoucí ústavu*

# Abstrakt

Tématem této bakalářské práce je automatizace Definition of Done v projektech vývoje softwarových aplikací. Práce vychází ze zásad agilní metodiky Scrum a její podstatou je zjednodušení vybraného procesu Definition of Done ve společnosti Kentico Software s.r.o. Na základě analýzy procesů DoD a možných automatizací byla vybrána a naimplementována statická analýza CSS souborů pomocí nástroje Stylelint. Výsledek této práce umožňuje automatickou kontrolu všech CSS souborů, čímž usnadňuje vývoj software a šetří čas vývojářům.



# Abstract

The topic of the thesis is the automation of Definition of Done (DoD) in software development application projects. The thesis builds upon the principles of an agile methodology named Scrum and its goal is to simplify the DoD process in the Kentico Software company. Based on the analysis of existing DoD processes and their possible automations, the author implemented an application for static analysis of CSS files using the Stylelint tool. The product of the thesis enables automatic checking of all CSS files and, as a result, streamlines software development in the company and saves developer time.



# Klíčová slova

Definition of Done, automatizace, Scrum, agilní vývoj, Stylelint, TeamCity, Kentico Software s.r.o.

# Keywords

Definition of Done, automation, Scrum, agile development, Stylelint, TeamCity, Kentico Software s.r.o.


# Citace

MERTA, Daniel. *Automatizace v projektech vývoje softwarových aplikací.* Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. RNDr. Jitka Kreslíková, CSc.

# Automatizace v projektech vývoje softwarových aplikací

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením paní doc. RNDr. Jitky Kreslíkové, CSc. Další informace mi poskytli Ing. Kamil Řezníček a Daniel Bruckner. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Daniel Merta
17. května 2017

## Poděkování

Za systematické vedení, poskytnuté rady a konzultace bych rád poděkoval vedoucí práce doc. RNDr. Jitce Kreslíkové, CSc. Také bych rád poděkoval kolegovi Ing. Kamilu Řezníčkovi za cenné informace týkající se možné automatizace v rámci Kentico Software s.r.o., dále pak rodině, kolegům a v neposlední řadě přítelkyni za trpělivost a projevenou podporu.

# Obsah

# Kapitola 1

# Úvod

Agilní metodika Scrum, která se používá ve firmě Kentico Software s.r.o. pro vývoj software, patří mezi nejčastěji používané agilní metodiky v České republice i ve světě. Tato metodika aplikuje základní přístupy agilního manifestu, uplatňuje specifické přístupy řízení vývoje a představuje alternativu k tradičním přístupům tvorby software.

Název metodiky vychází ze slova skrumáž, což je klíčová část celého procesu. Při tomto krátkém každodenním setkání se potkávají všichni členové týmu a referují zde o své činnosti z minulého dne, o tom, co budou v aktuálním dni dělat a také na jaké problémy narazili.

Vývoj prostřednictvím této metodiky probíhá v tzv. iteracích neboli sprintech, které obvykle trvají dva až čtyři týdny. Iterace jsou zakončeny setkáním všech zainteresovaných osob. Výsledkem každé iterace je funkční protopyp s úpravami vzniklými ve sprintu, který splňuje seznam výstupních kritérií (angl. „Definition of Done“, zkráceně DoD). Takovýto prototyp může být předveden zákazníkovi, který poskytne zpětnou vazbu. To umožňuje rychle a pružně reagovat na změny vznikající v průběhu projektu. Díky možnostem denně upravovat plán iterace je možné velice dobře reflektovat nové požadavky a rizika z nich vyplývající.

Seznam výstupních kritérií může být nejdůležitější kontrolní bod celého agilního projektu. Bez konzistentního seznamu totiž není možné odhadnout rychlost vývojového týmu. Takovýto seznam pomáhá zajistit, že každý prototyp má danou kvalitu, splňuje definované požadavky a obsahuje minimum chyb. Definici toho, co je hotovo, lze vnímat jako duši celého procesu Scrum.

Tato technická zpráva se skládá z několika částí. V kapitole 2 jsou vysvětleny základní pojmy, principy a možnosti DoD, z čeho vychází a k jakému účelu se používá. Konkrétním pravidlům, technikám a zásadám používaných ve společnosti Kentico Software s.r.o (zkráceně „Kentico“), se detailně věnuje kapitola 3. Kapitola 4 obsahuje analýzu procesů DoD, které je možné automatizovat v rámci vývoje v Kentico Software s.r.o. Na základě zjištěných automatizací byla po dohodě s konzultantem společnosti Kentico zvolena dílčí část DoD, jejíž návrh popisuje kapitola 5. Navržená automatizace a její implementace je popsána v kapitole 6 společně s demonstrací na vhodném vzorku dat. Poslední kapitola 7 pojednává o dosažených výsledcích a možnostech použití zvoleného postupu i pro další automatizaci.

V průběhu tvorby této práce bylo využito konzultací ve společnosti Kentico Software, s.r.o, která poskytla zdroje i materiály potřebné k analýze procesů DoD.

# Kapitola 2

# Definition of Done

Cílem této kapitoly je seznámit čtenáře se základními pojmy, které s Definition of Done souvisejí. Znalost níže uvedených pojmů je důležitá k získání širšího kontextu za jakým účelem a z jakého důvodu se DoD v agilních metodikách používá.

Je potřeba zmínit, že neexistuje žádný univertální DoD, který by pasoval na všechny typy možných projektů. Stejně tak může mít DoD různé podoby. Může existovat v digitální podobě, textové podobě na papíře, obou předchozích nebo jakékoliv jiné, která vyhovuje danému týmu a splňuje svůj účel, jímž je zlepšení kvality výsledného produktu a úspora času.

Scrum je model, který byl formálně definován na konci roku 1990, přesto trvalo několik let, než se začal v praxi uplatňovat. V současné době je však bezesporu nejčastěji používanou agilní metodikou vývoje v praxi.[19]

Jak bylo zmíněno v úvodní kapitole 1, vývoj probíhá v iteracích, kterým se říká **sprinty**. Rychlost týmu se měří pomocí počtu zpracovaných uživatelských story[1] v jedné iteraci, resp. sprintu. V praxi to vypadá tak, že v úvodních několika sprintech tým zkouší odhadnout, kolik story pointů je zhruba schopen dokončit. Díky tomu je možné lépe odhadnout výkonnost týmu a zvýšit tak přesnost plánování v dalších sprintech.

Rozlišujeme tři typy rolí, které se ve Scrumu vyskytují. Aby proces správně fungoval, je nutné, aby byla každá role zastoupena nejméně jedním uživatelem. Každá role má své specifické úkoly a požadavky, které popisuje podkapitola 2.1. V podkapitole 2.2 jsou popsány typy schůzek, které se hromadně označují za ceremonie. Poslední podkapitola 2.3 je pak věnována artefaktům, což jsou nástroje pro management ve Scrumu.

## 2.1 Uživatelské role

### Scrum master

Člověk na této pozici se stará o to, aby fungovala komunikace a mezilidské vztahy mezi jednotlivými členy týmu a mezi týmy samotnými. V literatuře je člověk s touto rolí často označován jako moderátor diskusí v týmu i mimo něj. Mimoto je zodpovědný za veškeré plánování a akce s tím spojené, organizuje schůzky, pohovory a společné sezení, ve kterých naslouchá potřebám členů týmu [18].

Také zajišťuje časový harmonogram jednotlivých akcí ve sprintu a stará se o celý tým. Je dobré, když má technický základ, aby mohl dobře porozumět a vést diskusi s technicky

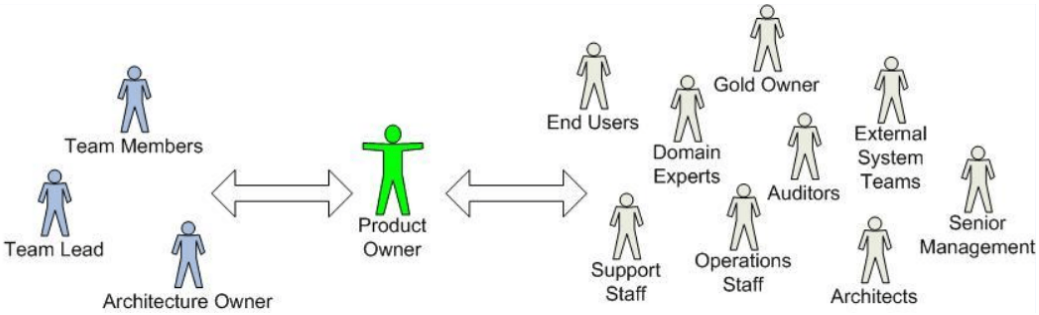
[1] Angl. „User stories“ - případy použití systému.

založenými členy týmu, avšak není to priorita. Hlavní je, aby takovýto člověk vzbuzoval důvěru v lidech, budoval dobré vztahy a měl měkké dovednosti (angl. „soft skills“) na skvělé úrovni. I když to nemusí být na první pohled zřejmé, jedná se o zásadní roli v celém procesu Scrum [20].

### Product owner

Tato osoba určuje pořadí, v jakém jsou jednotlivé user stories implementovány a také naslouchá potřebám zákazníků, aby mohl rozhodnout, co by bylo vhodné zahrnout do produktu a co ne [13]. Jinými slovy, definuje priority a rozhoduje, na čem se bude pracovat dříve, později, nebo vůbec. Pozice product ownera v týmu i mimo něj je vyjádřena obrázkem 2.1

Na rozdíl od scrum mastera nemusí sedět s konkrétním týmem v jedné místnosti, protože bývá často v kontaktu se zákazníky, avšak v případě potřeby je týmu k dispozici. Product owner musí znát velmi dobře vyvíjený produkt a mít jasno v tom, co se vyvíjí, aby se mohl správně rozhodovat.




Obrázek 2.1: Vizualizace role Product owner v procesu Scrum [5].

### Týmový člen

Do této role patří všichni ostatní členové týmu, kteří se podílejí na společném cíli. Jsou jimi například programátoři, testeři, designéři nebo analytici.

## 2.2 Typy schůzek

### Plánování sprintu

Této schůzky se účastní členové všech rolí z týmu a jak je patrné z názvu, probíhá před začátkem sprintu. Je vedena diskuse, při které se plánuje a odhaduje, co se ve sprintu stihne udělat. To znamená, že se vybrané user stories z product backlogu přemisťují do sprint backlogu tak, aby počet user stories odpovídal odhadnuté rychlosti daného týmu. Odhaduje se také časová náročnost jednotlivých user stories v tzv. user pointech, což jsou relativní jednotky toho, jak dlouho bude trvat zpracování. Doba věnovaná této schůzce, celkové diskusi a času pro odhadování, záleží na zkušenostech scrum mastera, který schůzi vede.

### Každodenní scrum

Jedná se o krátkou, několika minutovou schůzi na začátku pracovního dne, které se účastní členové týmu a scrum master. Členové týmu věcně sdělí, na čem aktuálně pracují a co mají v plánu ten den dělat. V případě jakýchkoliv vzniklých problémů je členové zmíní, aby o nich věděli všichni v týmu. Po uplynutí předem stanovené doby je úkolem scrum mastera schůzi ukončit i v případě, že nebyly všechny problémy vyřešeny. [17]

### Revize sprintu

V této schůzi jsou přítomny všechny role, koná se po ukončení sprintu a jde o ohlédnutí se za právě dokončeným sprintem. Cílem této schůze je zhodnotit, jestli vypracované user stories odpovídají původnímu odhadu a představě. Jestli ano, product owner user story uzavře a smaže z product backlogu. Jestliže se user story nepodařilo celou dokončit, původní se uzavře a vytvoří se nová, jejíž obsahem je právě nedodělaná část user story. V případě, že se nějaká user story nezačla vypracovávat vůbec, přesune se celá ze sprint backlogu zpátky do product backlogu. Tato schůze není nijak časově omezená a jejím cílem je komentovat dokončenou práci, případně k ní vznášet připomínky.

### Retrospektiva sprintu

Restrospektiva většinou plynule navazuje na revizi, takže se jí opět účastní všechny role. Účel této schůze spočívá v neustálé snaze zlepšovat práci a efektivitu celého týmu. V hromadné diskusi se řeší, co se povedlo, znovu se použije do budoucna a také čemu se v budoucnu vyhnout. Důležité je tedy vzít si zkušenosti, zamyslet se a vznést připomínky k právě dokončenému sprintu. [4]

## 2.3 Nástroje pro management

### Product backlog

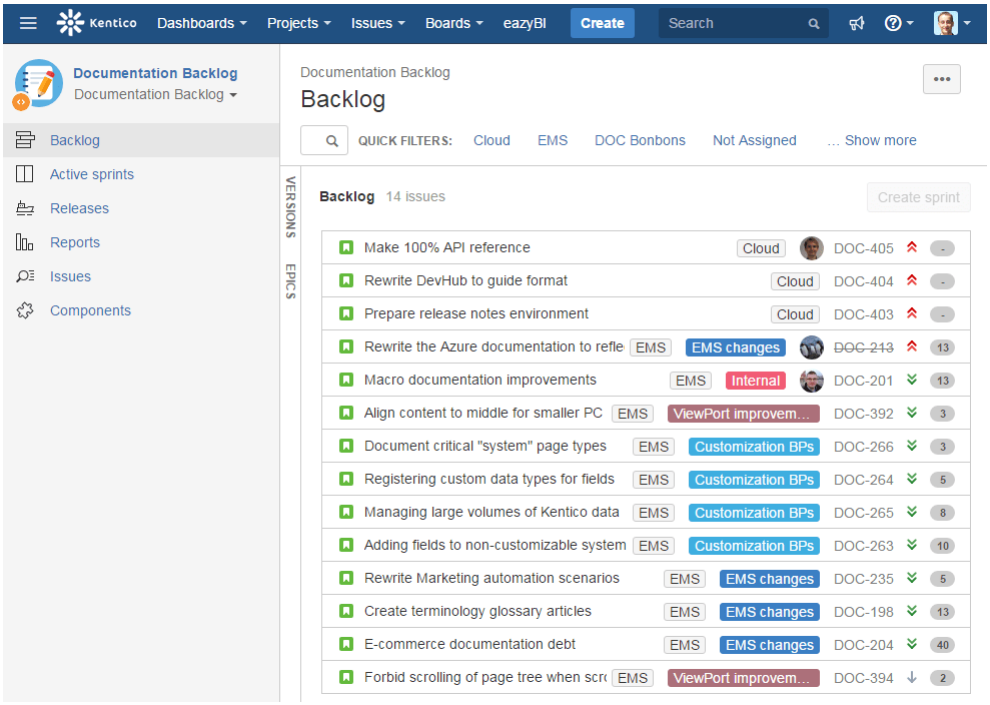
Product backlog je celkový prioritizovaný seznam user stories. Jedná se o formu úplné specifikace vyvíjeného softwaru. Nejčastějším a také nejjednodušším způsobem se backlog eviduje skutečně jako seznam či tabulka [21]. O backlog a jeho aktuálnost se stará product owner. V případě, že se nějaká user story nestihne dokončit ve sprintu, je možné ji vrátit zpátky do tohoto seznamu. Obrázek 2.2 je ukázkou toho, jak vypadá reálný backlog jednoho z týmů ve společnosti Kentico.

### Sprint backlog

Jedná se o podmnožinu product backlogu, která obsahuje user stories aktuálního sprintu. Tento seznam se tvoří v době plánování sprintu. Členové týmu si sami přiřazují jednotlivé user stories a po dokončení je označí jako hotové. Jestliže se nepodaří některé user stories dokončit za dobu sprintu, product owner je může buďto navrátit do product backlogu a nebo přesunout do nového sprint backlogu.

### Scrum board

Tabule pro scrum, neboli scrum board, může být jakákoliv tabule, stěna, skříň, nebo cokoliv jiného, co pomůže týmu zpřístupnit a zviditelnit položky sprint backlogu. Každý tým si



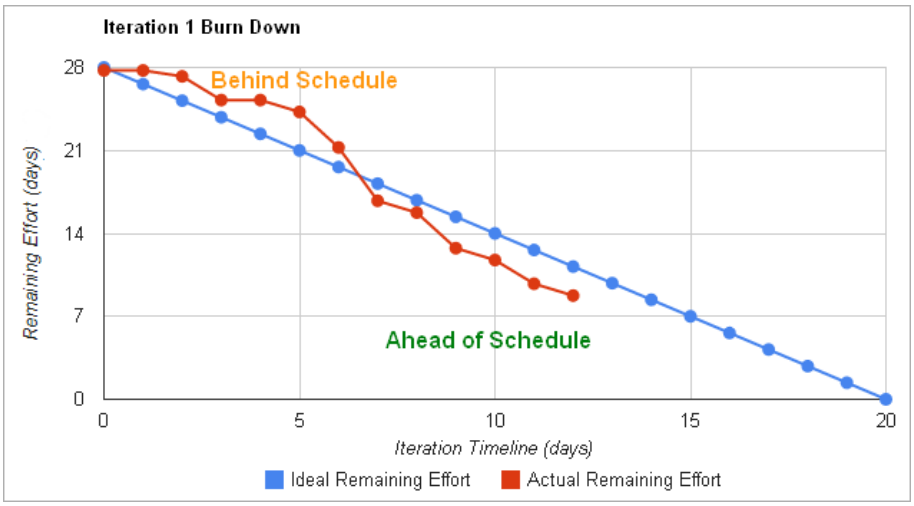

Obrázek 2.2: Skutečný backlog týmu z Kentico Software, s.r.o.

může tabuli přizpůsobit podle vlastních potřeb, tedy není důležité jak tabule vypadá nebo jaký má tvar. Hlavní je, aby na tabuli byly viditelné všechny položky pro aktuální sprint a jejich stav. O tuto tabuli se může starat kdokoliv z výše uvedených rolí.

### Burndown chart

Cílem tohoto grafu je znázornit rychlost zpracování user stories a vizualizovat tempo práce. Může sloužit také jako motivace pro členy týmu, kteří z něj snadno vyčtou, jestli je splněná práce ve sprintu kupředu, podle plánu, nebo jestli zaostává oproti původnímu plánu.

Ukázku takového grafu lze vidět na grafu 2.3, v němž osa X udává délku sprintu ve dnech a osa Y znázorňuje počet dní zbývajících do konce sprintu. Modrá čára je ideální plán vypracovávání user stories ve sprintu. Červená čára znázorňuje reálný průběh vykonané práce. Jestliže červená protne nulu na ose Y dříve než modrá čára, znamená to, že tým splnil naplánovanou práci dříve, než se očekávalo. Když červená čára nestihne protnout nulu na ose Y před skončením sprintu, tým nestihl vypracovat všechny naplánované user stories.

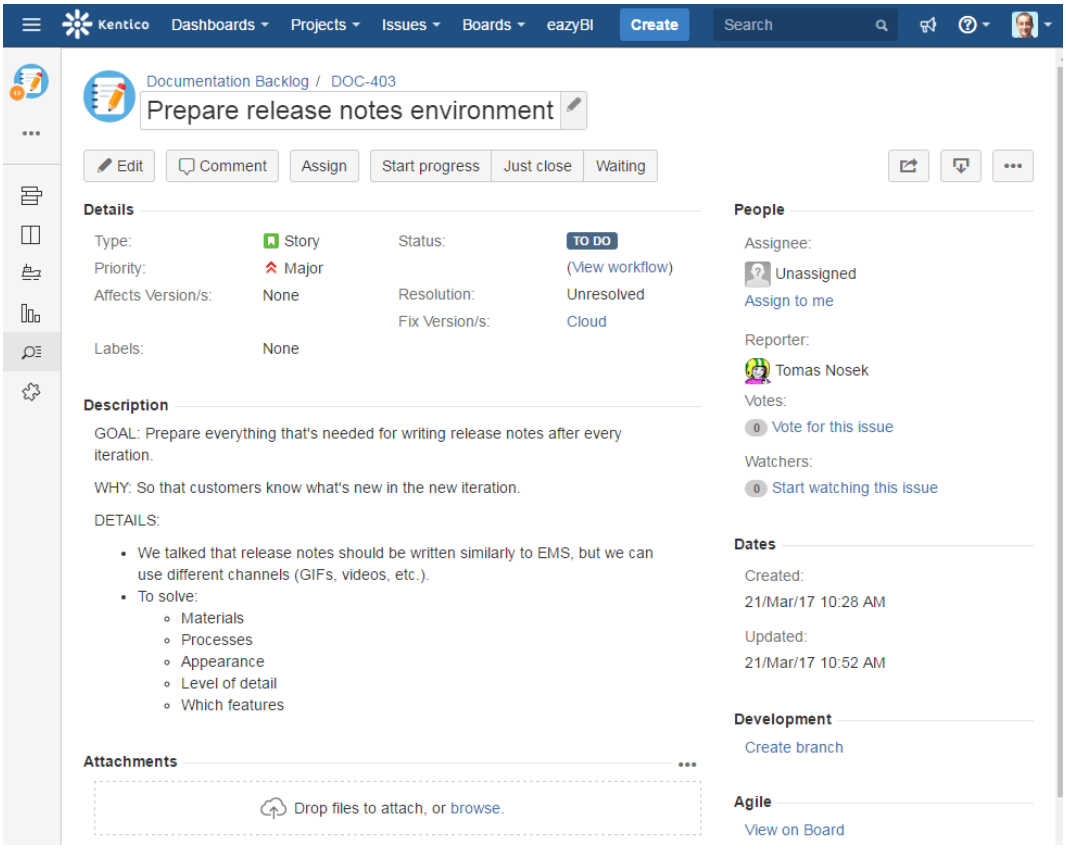



Obrázek 2.3: Ukázka základního Burndown chartu [8].

# Kapitola 3

# Definition of Done v Kentico Software s.r.o.

Ve firmě Kentico je technika Definition of Done používána pokaždé, když se končí uživatelská story. Tato kapitola shrnuje pravidla a zásady, které jsou ve story kontrolovány v závislosti na tom, z jakého oddělení pocházejí. Tento soubor pravidel, doporučení a technik lze vnímat jako pomůcku, která napomáhá nejen začínajícím zaměstnancům, ale hlavně jejíž dodržování zaručuje jasně definovanou úroveň a kvalitu výsledného produktu. Na obrázku 3.1 je ukázková user story, v níž lze vidět jednotlivé atributy, které ji charakterizují.




Obrázek 3.1: Ukázka zápisu user story v systému.

## 3.1 Architektura

Základním kamenem úspěchu pro komerční software typu Kentico CMS je vhodně navržená architektura. Zákazníci mají možnost zakoupit si i licenci na zdrojový kód, ve kterém mohou dělat libovolné změny a upravovat dílčí fragmenty podle vlastních potřeb. I proto je samotnému kódu a jeho kvalitě věnována značná pozornost, neboť firmu z tohoto pohledu reprezentuje.

Dalším kardinálním důvodem je znovupoužitelnost kódu. Vždy ke konci roku se po roční iteraci vydává nová verze produktu. Je tudíž žádoucí, aby byl stávající i nově vznikající kód dobře čitelný, logicky navržený a mohli se v něm snadno vyznat i programátoři, kteří jej nepsali. Dodržování těchto pravidel dokáže ušetřit nemalé množství času při každé iteraci, jelikož programátoři neztrácejí zbytečně čas zkoumáním již existujících řádků kódu. V této souvislosti je veškerý nově napsaný kód před jeho zahrnutím do aktuální stabilní verze programu kontrolován jiným programátorem na seniorské pozici, který v případě nesrovnalostí na tyto programátora upozorní.

V rámci architektury programátoři dodržují detailní pokyny pro psaní kódu a také osvědčené postupy, které se za dobu vývoje osvědčily. Následují ukázky z jednotlivých programovacích jazyků a případů, které se při vývoji Kentico CMS používají:

### 3.1.1 Pokyny pro psaní kódu

**C#** - konvence pro pojmenování jsou implementovány jako sada pravidel v ReSharper[1] konfiguračním souboru. V něm jsou obsažena pravidla pro každý typ identifikátoru a vycházejí z konvencí společnosti Microsoft. Ze samotného názvu by měl jít vyčíst význam, přičemž se nepoužívají zkratky, podtrhávací čáry, pomlčky ani jiné nealfanumerické znaky. Název funkce by tedy mohl být například `GetWindow` namísto `GetWin` či `Get_Window`. K rozlišení slov v názvech identifikátorů se používají velká písmena tak, že buďto začíná velkým písmenem každé slovo v identifikátoru, např. `PropertyDescriptor`, nebo začíná velkým písmenem každé slovo v identifikátoru, kromě prvního: `propertyDescriptor`. Z hlediska stylu psaní je doporučeno vyhnout se regionům a místo nich používat třídy, aplikovat klíčová slova pro vestavěné C# typy, vkládat otevírací závorku jen po posledním příkazu `using` a nepoužívat ternární operátory.

**ASP.NET** - pro pojmenovávání zde platí stejné konvence jako v programovacím jazyce C#. Každý ovládací prvek Kentica CMS je označen prefixem *cms*:
`<cms:TextBox ID="txtFirstName"runat="server"/>`.
Prefixem *ug* pak začínají UniGrid[2] ovládací prvky v šablonách:
`<ug:Action Name="edit"/>`.

**Javascript** - k vytvoření objektů se používá doslovná syntaxe a zároveň se nepoužívají rezervovaná slova [7] pro názvy klíčů, neboť nefungují v prohlížeči Internet Explorer verze 8. Ukázka špatně vytvořeného objektu:

```
var item = new Object();

var superman = {
```

[1] ReSharper - rozšíření pro vývojové prostředí Visual Studio sloužící ke zlepšení produktivity a refaktorizaci kódu.

[2] UniGrid - ovládací prvek umožňující zobrazení dat ve vysoce přizpůsobitelné a flexibilní tabulce.

```
    default: { clark: 'kent' },
    private: true
};
```

Správně vytvořený objekt:

```
var item = {};

var superman = {
    defaults: { clark: 'kent' },
    hidden: true
};
```

V případě potřeby zkopírování pole se nepoužívá cyklus, který kopíruje jednotlivé prvky pole:

```
var len = items.length,
    itemsCopy = [],
    i;

for (i = 0; i < len; i++){
    itemsCopy[i] = items[i];
}
```

ale metoda `Array.slice`:

```
var len = items.length,
    itemsCopy = [],
    i;

itemsCopy = items.slice();
```

Pro převod objektu podobného poli (angl. „array-like object“) na pole se opět používá metoda `Array.slice`:

```
trigger = function () {
    var args = Array.prototype.slice.call(arguments);
    ...
}
```

Tečková notace se využívá pro přístup k vlastnostem:

```
var luke = {
    jedi: true,
    age: 28
};

var isJedi = luke.jedi;
```

Přístup k vlastnostem s proměnnou je zajištěn indexovou notací:

```
var getProp = function (prop) {
    return luke[prop];
}
```

```
isJedi = getProp('jedi');
```

**CSS** - Všechny ukázkové webové stránky musí být validní ve formátu CSS 3.0. Z toho důvodu musí být každý CSS soubor zvalidován a otestován ve všech podporovaných webových prohlížečích.

Je-li vyžadována podpora jazyků, které se čtou zprava doleva (angl. „right-to-left language“), je nutné, aby všechny třídy, které obsahují nějaké specifické zarování nebo formátování, byly přepsány pro tento specifický jazyk a tyto atributy zrcadlily. Tedy kromě standardní definice třídy:

```
.message-info {
     padding-left: 10px;
}
```

je zapotřebí tuto třídu přepsat i pro jazyk, který se čte opačně:

```
.message-info {
     padding-left: 10px;
}

.RTL .message-info {
     padding-left: auto;
     padding-right: 10px;
}
```

Pro všechny moderní webové prohlížeče je doporučeno používat prefixy, které jsou stanoveny jejich prodejci. Následuje ukázka s prefixem prohlížece Mozilla („moz“) a Chrome („webkit“):

```
.message-info {
     -moz-border-radius: .5em;
     -webkit-border-status: .5em;
     border-radius: .5em;
}
```

Jediný specifický je v tomto ohledu prohlížeč Internet Explorer, pro něhož se musí odlišné styly definovat v samostané rodičovské třídě:

```
.message-info {
     color: blue;
}

.IE8 .message-info {
     color: green;
}
```

V ASPX nebo HTML kódu se smí používat pouze třídy z CSS souboru namísto vložených stylů. Příkaz `!important`, který zvyšuje důležitost CSS hodnot, je doporučeno používat pouze výjimečně, nebo ideálně vůbec.

**API** - Dokumentace aplikačního programovacího rozhraní, zkráceně API[3], se skládá z XML tagů umístěných ve zdrojovém kódu Kentica CMS ve formě komentářů. Platí zásada, že kód je v konečném důsledku čten vícekrát, než-li je psán. Proto je kladen důraz na to, aby dokumentace API byla kvalitní, díky čemuž mohou vývojáři pracovat s API Kentica CMS efektivně. Obecně platí:

- Pojmenování samotných členů a tříd by mělo být výstižné.
- API dokumentace by měla být co nejjednodušší a nejkratší.
- Je třeba dávat obvzvlášť pozor na komentáře při refaktorizaci existujícího kódu.
- Pro zmínění dalších členů API jsou použity odkazy ve formátu `<see cref = "Member»`.
- Všechny vlastní komentáře se musí zkonzultovat s příslušnou osobou v týmu.
- Komentáře jsou napsány v přítomném čase.
- Používá se americká angličtina namísto britské.

**Výjimky** - vlastní výjimky se navrhují podle šablony společnosti Microsoft [2]. Používají se pouze standardní typy výjimek [14] specifikované pro platformu .NET.

Výjimky musí být také serializovatelné, aby fungovaly správně v celé aplikační doméně a vzdálených hranicích. Pokud je to možné, nepoužívají se výjimky pro normální tok řízení. Použití výjimek pro řízení toku porušuje princip minimálního překvapení (angl. „least astonishment“), což ztěžuje čitelnost programu.

Nepoužívá se formát `[ClassName.MethodName]` v rámci výjimek, jelikož jde o redundantní informaci, která zvyšuje údržbu celé aplikace, protože hodnota by musela být změněna po přejmenování nebo přesunutí dané třídy či metody.

**Porovnání řetězců** - vždy je doporučeno používat přetížení metod, které akceptují argument `StringComparison`. Díky tomu je možné deklarovat, jak bude zpracování textových řetězců probíhat. Aby byly všechny položky z kolekcí normalizovány na stejnou velikost znaků (buďto malá, nebo velká písmena), je zapotřebí použít kolekci konstruktorů, která akceptuje znak porovnání jako argument.

Nepoužívá se ordinální srovnání řetězců, které mohou obsahovat uživatelská data. Metody **CSafe* je zakázno použít, jelikož jsou nestandardní. Stejně tak metoda *ToLower* a *ToUpper* je nepotřebná, protože porovnávací metody obecně ignorují velikost znaků.

**SQL** - názvy indexů jsou ve formátu:

`IX_<nazev_tabulky>_ <sloupec1>_<sloupec2>...<sloupecN>`. Tedy například název

`IX_Analytics_Statistics_StatisticsCode_StatisticsSiteID_StatisticsObjID` značí index nad tabulkou `Analytics_Statistics` s indexovanými sloupci ve stejném pořadí: `StatisticsCode, StatisticsSiteID, StatisticsObjsID`.

Index nad primárním klíčem je ve formátu `PK_<nazev_tabulky>`, například

`PK_Analytics_statistics`. Indexů by nemělo být moc a tam, kde se typicky používá několik sloupců najednou, by měl být index složený z několika sloupců tak, aby byly v pořadí od obecnějšího k méně obecnému.

[3] Application Programming Interface (API) označuje rozhraní pro programování aplikací.

### 3.1.2 Osvědčené postupy

Při vývoji se dbá na to, že nedochází k porušení následujících pravidel:

- Zdrojové soubory jsou ukládány v sestavách daných modulů.
- Objekty vložené do relace nebo mezipaměti jsou serializovatelné.
- Musí být zřejmé, jaký způsob synchronizace objekty podporují.
- Administrační stránky jsou založené na šablonách uživatelského rozhraní.
- Jsou podporovány webové farmy.
- Komentáře k API dodržují pokyny dokumentace API.
- Změny v API jsou vyřešeny v porovnávacím nástroji k tomu určeném a také jsou součástí novější verze.

Pro správu zdrojového kódu mezi týmy se v Kenticu používá Team Foundation Server[4]. Před nahráním pozměněného kódu do sdíleného repozitáře je vhodné stáhnout si poslední verzi sestavení a přeložit ji. V případě, že se objeví nějaké upozornění, je zodpovědností daného týmu, který modul vlastní, aby upozornění řešil.

Jestliže se vývojář rozhodne provést podstatné změny ve zdrojovém kódu, musí tyto změny předem konzultovat s příslušným vlastníkem daného modulu a jednat podle domluvy.

Při testování rozlišujeme dva základní typy testů, unit a integrační:

1. Unit testy jsou zaměřené na testování izolované části aplikace, což může být metoda stejně jako instance třídy. Důležitým předpokladem je, že metoda nebo třída nemá skryté závislosti, protože jen tak může být izolována a testováno pouze požadované chování.
2. Integrační testy se naopak zaměřují na testování spolupráce více částí aplikace na jednom úkolu. Výborně se tak hodí k testování funkčních požadavků, tedy ověření, zda aplikace dělá, co má. Databáze je z pohledu testů jen další závislost, kterou je možné v některých případech izolovat.

U databázových změn je důležité pohlídat si, že:

- Testovací objekty jsou pojmenovány s prefixem „text“ + „loginName“.
- Šablony stránek se klonují a používají se tzv. *ad-hoc šablony.*
- Pokud je to možné, nemění se hodnoty na globální úrovni, ale raději na úrovni jednotlivých webů.
- Změny v databázi, které se mají promítnout do produkčního řešení, je zapotřebí vyřešit co nejdříve, aby nedocházelo ke zdržení v jiných týmech.
- U novějších verzí je třeba brát v úvahu pořadí skriptů a skripty označené jako „last“ nechat na konci.
- Případy použití systému se testují i na novějších verzích aplikace.

[4]Team Foundation Server (zkráceně TFS) od společnosti Microsoft nabízí komplexní řešení pro týmový vývoj aplikací.

## 3.2 Dokumentace

Dokumentace je základní dokument, ze kterého vychází drtivá většina uživatelů, kteří přijdou do styku s firemním produktem. Celá dokumentace produktu Kentico CMS[5] je psána v anglickém jazyce, neboť je produkt určen zákazníkům z celého světa. Dokumentace musí být přesná, kompletní, jednoznačná a co nejvíce výstižná pro cílovou skupinu čtenářů. Tuto činnost mají na starosti dokumentaristé, kteří vytváří a modifikují technickou dokumentaci jak pro interní, tak pro externí účely. Interní dokumentace slouží zejména jako zdroj informací pro vývojáře a technickou podporu, přičemž jsou zde uvedeny limity, detaily a jiné neveřejné informace. Při psaní externí dokumentace je důležité zohledňovat cílovou skupinu uživatelů, kteří budou dokument nejčastěji využívat. Možní uživatelé, na které bývá dokumentace zaměřena, jsou:

1. běžný koncový uživatel - reprezentuje uživatele bez hlubších počítačových znalostí, který se zvládá pohybovat po internetu a užívat základní počítačové nástroje. Typicky zde patří editor obsahu, obchodník nebo vedoucí internetového obchodu. Každá jednotlivá akce se rozepisuje jako samostatný krok, aby nedošlo ke zmatení uživatele. Nepoužívá se žádná speciální terminologie, ale naopak je zde použito větší množství obrázků pro názornost.

2. pokročilý uživatel - reprezentuje uživatele s hlubšími znalostmi o tom, jak nakonfigurovat nebo rozvíjet systém pro správu webového obsahu. Předpokládá se, že uživatel má alespoň základní programátorské dovednosti a znalost terminologie. Takovéto znalosti lze nabýt jak z Kentico CMS, tak i z mnoha dalších programů, které jsou určeny pro správu webového obsahu. Je zde možné sloučit více jednodušších akcí do jediného kroku a také používat pojmy z oblasti informačních technologií.

V aktuální verzi dokumentace se lze setkat se dvěma přístupy, jakými byly dokumenty vytvořeny:

1. orientovaný funkčností - v minulosti hojně používaný přístup pro popis uživatelského rozhraní a všech tlačítek či elementů v něm. Je vhodný pro popis funkcionality uživatelského rozhraní méně zdatným uživatelům. Tyto dokumenty popisují funkčnost, nikoliv konkrétní scénáře. Z toho důvodu je množství takovýchto dokumentů stále snižováno.

2. orientovaný úkolem - popisuje, čeho může uživatel s danou funkcí dosáhnout a jaké scénáře má k dispozici. Aktuálně je to více využívaný přístup, jelikož uživatelé upřednostňují celkový postup, jak nějaké funkčnosti dosáhnout, nebo jak ji nakonfigurovat, namísto popisu všech elementů v jisté sekci uživatelského rozhraní.

Každý článek v dokumentaci se řídí hlavním pravidlem - musí být jednoznačně rozlišitelný ve stromovém obsahu a také nezávislý vůči jiným dokumentům. Pokud je zapotřebí, aby si uživatel přečetl i jiné související články, musí na ně být v aktuálním článku odkaz. Dokumenty, zejména ty úkolem orientované, jsou složeny ze čtyř částí:

### 3.2.1 Titulek

Všechny nadpisy jsou psány v přítomném průběhovém čase a musí obsahovat významové sloveso s příponou *-ing*. Výjimkou, kdy nadpis nemusí obsahovat sloveso, jsou stránky,

[5] Content Management System (CMS) - software pro správu webového obsahu.

které slouží jako rozcestník pro objasnění podřízených stránek. Typický příklad nadpisu s významovým slovesem je titulek *Creating pages*. Příkladem rozcestníkové stránky je titulek *Pages*.

### 3.2.2 Úvod

V úvodu bývají obvykle vysvětleny otázky, které pomáhají uživateli v tom, že ví, co má kde, kdy a jak použít:

- Co funkce dělá?
- Proč funkci použít?
- Kdy funkci použít?
- Jaké jsou prerekvizity a čím začít? (volitelná)
- Kdo by měl funkci použít? (volitelná)
- Kde funkci použít? (volitelná)

### 3.2.3 Tělo

V těle jsou popsány jednotlivé kroky, které je třeba vykonat. Když záleží na pořadí kroků, použije se číslovaný seznam, v opačném případě stačí odrážky. Je vhodné dbát na to, aby kroků nebylo příliš moc - eliminuje se tak šance, že uživatel udělá chybu. Pro jazyk textu se typicky používá imperativ s modálními slovesy. Typická slovesa pro interakci jsou: *type, press, click, double-click, right-click* a *select*.

Dokumentarista musí dodržovat typografické zásady typické pro Kentico (nepoužívání dlouhé pomlčky, znaku mínus, šikmých uvozovek, atd.) a také správnou terminologii (jednoduchá a jasná americká angličtina, správný formát slov, nepoužívání zkrácených tvarů atd.).

Obrázky se vkládají pouze tam, kde to dává smysl a uživateli to pomůže pochopit problém. V Kenticu se používá buďto program FastStone Capture (odkaz?) nebo Greenshot. Všechna privátní data na nich musí být rozostřena a kritické části obtáhnuty červeným obdélníkem.

### 3.2.4 Výsledek

V této části je popsáno, co se stane, když uživatel vykoná předešlé kroky. Rovněž je zde uvedeno, jak si uživatel ověří, že je funkčnost nastavena správně, případně jak se má co chovat.

Speciální částí dokumentace jsou **poznámky k vydání** a také **instrukce pro přechod na novější verzi**. Tyto články se píší těsně před vydáním nové verze produktu a obsahují popis nových funkcionalit, změn v dosavadním produktu a také seznam oprav.

## 3.3 Uživatelské rozhraní

K tvorbě uživatelského rozhraní se v Kentico využívá především programovací jazyk LESS, CSS a Javascript. Pro tyto jazyky mají programátoři k dispozici směrnice, ve kterých jsou sepsána pravidla a konvence vhodného stylu programování.

### LESS

LESS je dynamický jazyk pro tvorbu stylů. Rozšiřuje starší, avšak stále více užívaný jazyk CSS o dynamické prvky typu proměnné, funkce a operace. V mnoha případech jsou i tak směrnice pro CSS a LESS shodné. Kromě definic, v jakém tvaru má být jméno proměnné, kdy používat komentáře či jaké barvy lze použít, je doporučeno používat lokální pracovní stanici, aby bylo možné pracovat na jednom souboru simultánně. Pro opravování chyb platí, že se prvně zkouší stávající kód odebrat, nežli je přidán další. Příklad špatné definice vlastností:

```
color: #262524;
font-size: 14px;
```

Správná specifikace vlastností je pomocí proměnných:

```
color: @text-color;
font-size: @font-size-base;
```

### CSS

Pro pojmenování tříd v CSS se používají výhradně malá písmena a u víceslovných názvů jsou slova oddělena pomlčkou. Třídy jsou od sebe odděleny prázdným řádkem a začínající závorka je vždy na stejném řádku, jako název třídy. Nesmí být zapomenuto na mezeru mezi názvem třídy a začínající závorkou, jelikož to zlepšuje čitelnost kódu. Další dvě nezbytná pravidla jsou středník po ukončení deklarace a mezera za dvojtečkou v názvu vlastnosti. Příklad nesprávného zápisu barev:

```
.message-info {
     color: rgb(0, 0, 0);
     background: #FFFFFF;
     border-color: black;
}
```

Správná definice barev je v hashovém formátu, malými písmeny a pokud možno ve zkrácené verzi:

```
.message-info {
     color: #000;
     background: #fff;
     border-color: #000;
}
```

### Javascript

Na začátku všech javascriptových modulů musí být použit příkaz `use strict` [11], jehož účelem je, aby se kód vykonal v tzv. striktním režimu.. Je žádané, aby byl kód modulární, neboť je pak zajištěno, že je případný kód zákazníka oddělen od nativního kódu v pozadí. Dalším pravidlem je, aby nebyl nikdy použit globální stav, protože jen tak lze udržet moduly na sobě nezávislé a soběstačné. S ohledem na výkon výsledného skriptu je při psaní kódu snaha o minimalizaci Document Object Model (zkráceně DOM) [10] dotazů a dalších překreslovacích technik. DOM dotazy totiž navrací první element v dokumentu, který se shoduje s hledaným a nejsou příliš efektivní. Ukázka nesprávné deklarace proměnných:

```
var items = getItems();
```

```
var goSportsTeam = true;
var dragonball = 'z';
```

Vhodné je pouze jednou deklarovat `var` a každou proměnnou na nový řádek:

```
var items = getItems();
    goSportsTeam = true;
    dragonball = 'z';
```

Mimo výše zmíněná pravidla se také kontroluje, že kód obsahuje pouze kaskádové styly, nikoliv styly vložené do řádku. HyperText Markup Language (HTML) kód musí být validní s formátem HTML5, což se ověřuje přes webovou službu Markup Validation Service (W3C). Stránky, které používají některý ze zmiňovaných kódů, je potřeba otestovat ve všech podporovaných prohlížečích.

## 3.4 Legálnost

U každé komponenty třetí strany je nutné ověřit kompatibilnost licence a také, že splňuje dílčí licenční požadavky. V případě, že je komponenta licencována pod více než jednou licencí, alespoň jedna z licencí musí být uvedena v seznamu povolených licencí. Je povinností toho, kdo jakoukoliv novou komponentu do produktu vkládá, aby splnil všechny nutné podmínky použití dané licence. Povolené licence a jejich podmínky užití v rámci Kentico CMS jsou:

- LGPL (angl. „Lesser General Public license“) - komponenty lze používat bez omezení. Pouze v případě změny ve zdrojovém kódu je povinnost dát takto změněný kód k dispozici k veřejnému stažení.
- MIT, nebo též X11, dvoubodová BSD (angl. „Berkeley Software Distribution“ a Apache licence - komponenty lze používat a modifikovat bez omezení, avšak text licence musí být uveden v instalátoru softwaru.
- MPL (angl. „Mozilla Public Licence“) - komponenta smí být použita, ale je zapotřebí zveřejnit její zdrojový kód v souladu s globálními podmínkami této licence.
- WTFPL (angl. „do What The F*ck you want to Public License“) - komponenty s touto licencí je možno užívat bez jakýchkoliv omezení

Naopak je nutné vyhnout se použití komponent s licencí GPL (angl. „General Public License“), neboť dle podmínek této licence by nebylo možné produkt prodávat a musel by být zdarma věřejně dostupný.

## 3.5 Optimalizace výkonu

U takto robustního softwaru pro správu webového obsahu je obvzlášť důležité myslet na optimalizaci výkonu. V modulech, ve kterých to dává smysl, se měří a testuje požadovaná rychlost a také mezní dosažitelné hodnoty. Platí také pravidlo, že novější verze dané funkce nesmí být méně výkonná, nežli verze starší. Například marketingová automatizace musí zvládnout zpracovat alespoň 10 000 kontaktů, e-mailový marketing musí zvládnout odeslat minimálně 100 000 e-mailů za 24 hodin a správa kontaktů musí umět pracovat s alespoň

100 000 kontakty a s 1 000 000 aktivit. V případě nedodržení těchto minimálních hodnot se musí modul zrevidovat a kód optimalizovat.

## 3.6 Ukázkové webové stránky

Každá funkcionalita Kentica CMS je demonstrována na některé z ukázkových webových stránek. Po implementaci či úpravě dané funkčnosti je nutné překontrolovat, že daná funkce pracuje správně, bez chyb a že je možné ji použít pro demonstrační účely.

## 3.7 Bezpečnost

Pravidlům z oblasti bezpečnosti je věnována obvzlášť velká pozornost při každém sprintu[6]. Případná zranitelnost v produktu by totiž mohla mít enormní následky nejen pro zákazníka, ale také pro celé Kentico. Z tohoto důvodu musí každý člen týmu absolvovat jednou do roka bezpečnostní školení.

### Databáze

K prevenci SQL injection[7] je zapotřebí ošetřovat všechny přístupy k databázi tak, aby útočník neměl možnost podsunout svůj pozměněný SQL dotaz k databázi. V uložených procedurách je tak zakázáno použití příkazu Execute[8]. Rovněž všechny přístupy k databázi probíhají formou parametrizovaných dotazů[9].

### Kód

Při psaní kódu je třeba dodržovat několik obecně známých bezpečnostních pravidel. K zabránění XSS[10] se kontroluje, že registrované skripty neobsahují žádný nebezpečný vstup a také, že uživatelský vstup je správně oddělen od důležitých částí programu. Uživatelský vstup bývá často nejzranitelnější částí systému. Proto se u všech vstupů, skrytých polí a cookies kontroluje datový typ, formát dat, délka a rozsah. Takovéto kontroly neprobíhají na straně klienta, ale jsou implementovány přímo na serveru.

Také je třeba zabránit vystavení interních implementací objektů uživatelům (angl. „Insecure Direct Object Reference“). Proto jsou metody pro požadavek a odezvu (angl. „GET“ a „POST“) mezi klientem a serverem vždy validovány.

Obranu proti CSRF[11] útoku zajišťuje v kritických částech metoda POST. Je také zakázáno externí či neověřené přesměrování.

[6] Sprint, neboli iterace, je základní jednotka vývoje pomocí agilní metody Scrum. Doba sprintu je určena předem v rozmězí od jednoho týdne po jeden měsíc.

[7] SQL injection - technika napadení databázové vsrstvy vsunutím kódu přes neošetřený vstup a vykonání vlastního pozměněného SQL dotazu.

[8] Execute - příkaz, který vykoná příkazový řetězec nebo řetězec znaků v rámci transakční SQL dávky nebo v rámci uložené procedury.

[9] Parameterized query - dotaz, ve kterém se používají zástupné symboly pro parametry a hodnoty parametrů jsou dodány až v době provádění.

[10] XSS (Cross-site scripting) - metoda narušení webových stránek využitím bezpečnostních chyb ve skriptech, především pomocí neošetřených vstupů.

[11] CSRF (Cross-site Request Forgery) - metoda útoku na webovou aplikaci pracující na bázi nezamýšleného požadavku pro vykonání určité akce z nelegitimního zdroje.

### Přihlašování a šifrování

Všechny aktivity, které jsou z nějaké důvodu kritické či důležité, jsou logovány. V aplikaci nesmí existovat výjimka, která by nebyla ošetřená. Šifrování je použito všude tam, kde jsou použity přístupové údaje, klíče či hesla.

### Výpadek stránek

Je vhodné dopředu zvážit scénáře, ve kterých by mohlo dojít k výpadkům a nedostupnosti stránek. DDoS útok[12] je mezi kyber-zločinci nejrozšířenější technika, avšak obrana proti ní prakticky neexistuje. Vhodným použitím maker a zástupných znaků (wildcards = poznámka pod čarou) však můžeme riziko takového útoku snížit.

### Souborový systém

Kromě databáze bývají citlivá data uložena i v souborovém systému. Ten se tak stává hned po databázi druhým nejčastěji napadeným prvkem. Proto se zde ošetřuje, že uživatel má pro přistoupení k zabezpečenému souboru dostatečná práva.

## 3.8 Testování

K automatizovanému testování se používá volně dostupný aplikační rámec pro .NET - *NUnit*. Každý test musí být zařazen do příslušné kategorie, která daný typ testu reprezentuje. V testech nesmí být použita klíčová slova jako *if, switch, for, while, try-catch* nebo *break* a zároveň musí všechny testy skončit úspěšně bez vyvolání chyb. Výjimku mají pouze testy, které pokrývají funkcionalitu, která ještě není naimplementována. V tom případě musí být dočasně označeny atributem *Ignore*. I testy samotné procházejí procesem schvalování a testování.

Je dobré vyhnout se používání opakovaných částí kódu při testování. K testování kombinací vstupů a výstupů se používají parametrizované testy. Pro kód, který je vykonán před nebo po testu v testovací sestavě se používají metody *SetUp* a *TearDown*. Pokud je stejná základní třída vyžadována v různých testovacích projektech, musí být umístěna v projektu **.Base.Tests*.

Samotné testy se nesmějí ovlivňovat a nesmí v nich být použity atributy

`RegisterObjectType` či `RegisterDocumentType`. Aby se zabránilo přístupu k souborovému systému při testování, používají se vestavěné prostředky nebo virtuální souborový systém.

NUnit nabízí bohaté množství tvrzení (angl. „assertions“ ve formě statických metod. K dispozici jsou tvrzení pro kontrolu rovnosti, identity, podmínky, typu, porovnání, výjimky a jiné.

## 3.9 MVC

Model-View-Controller je specifická architektura webových aplikací, která rozděluje datový model aplikace, uživatelské rozhraní a řídící logiku do tří nezávislých komponent. Kentico nepodporuje v MVC aplikaci všechny funkce, které jsou standardně dostupné při zakoupení Kentico CMS. Nicméně pro funkce, které podporovány jsou, platí následující pravidla:

[12] DDoS (Distribute Denial of Service) - přehlcení cílové služby požadavky

- Funkcionalita je otestována a demonstrována na ukázkové webové stránce.
- Funkcionalita je otestována v prostředí webových farem (angl. „Web Farms“) a Microsoft Azure[13].
- Každá funkčnost musí být pokryta testem.
- Testovací scénáře nejsou určeny pouze testerům, ale také členům ostatních vývojových týmů.
- Pro každou funkci v MVC musí existovat samostatný článek v dokumentaci. Jestliže se popis funkčnosti v MVC nelíší od standardní dokumentace pro Kentico CMS, vytvoří se alespoň stránka s odkazem na originální dokumentaci.
- Funkcionalita MVC aplikace by neměla být ovlivněna nastavením a vlastnostmi uvnitř uživatelského rozhraní Kentica. Jestliže to situace vyžaduje, je zapotřebí to důkladně zvážit.
- Každá MVC story je na svém konci zkontrolována vývojářem z jiného tým.

[13]Microsoft Azure - aplikační platforma využívající cloud a datacentra Microsoftu k hostování a běhu aplikací

# Kapitola 4

# Analýza

Analýza procesů DoD a jejich možná automatizace je stěžejním bodem této práce. V této kapitole jsou popsány procesy, u kterých automatizace dává smysl a které jsem identifikoval na konzultacích s vedoucími pracovníky dílčích sekcí.

## 4.1 Pravidla pro kontrolu kódu

Pro kompilaci Kentico CMS je využíván překladač Roslyn. K tomuto překladači je možné naprogramovat pravidla v jazyce C#, pomocí kterých bude v době kompilace kontrolován kód. V současné době musí testeři překompilovaný kód testovat statickými testy, z nichž ne všechny jsou automatizované. Kdyby existovala vhodná pravidla pro kontrolu všech částí kódu již při kompilaci, odpadla by potřebova testovat kód statickými testy. Je téměř nemožné vytvořit kontrolní pravidla pro celé řešení, avšak je možné naprogramovat pravidla pro jisté soubory či logické části Kentico CMS.

## 4.2 Generování kapitol z dokumentace do PDF souborů

Při vydání každé nové verze Kentica je zapotřebí vygenerovat jednotlivé sekce dokumentace do PDF souborů, které jsou poté uživatelům dostupné ke stažení pro případ, že by je někdo potřeboval využívat při nemožnosti přístupu k internetu. V současné době se toto provádí manuálně pomocí nástroje pro převod do PDF souboru. Celá dokumentace je uložena v cloudovém řešení společnosti Atlassian, která má k dispozici vlastní Application Programming Interface (API). Pomocí volání Representational State Transfer (REST) by bylo možné jednotlivé sekce stáhnout a poté je vhodným programem převést do PDF formátu bez nutnosti manuálního zásahu.

## 4.3 Statická analýza CSS souborů

Ke kontrole správnosti CSS souborů je využíván program CSSLint. Dosud je k této analýze zapotřebí programátor, neboť proces není automatizovaný. CSSLint má také veřejně dostupné API, pomocí kterého by bylo možné vytvořit program, který by tuto analýzu prováděl automaticky a výstup z ní uložil do souboru. Existují však i jiné nástroje, které je možno integrovat do testovacího prostředí používaného v Kenticu, které sdružuje všechny ostatní testy pro DoD.

## 4.4 Validnost zobrazení stránek pomocí BrowserStack

BrowserStack je nástroj, který umožňuje simulovat zobrazení jednotlivých webových stránek ve většině existujících webových prohlížečích. V Kentico CMS je tento program využíván pro kontrolu správnosti zobrazení stránek na ukázkových webech. Aktuálně je pro tuto kontrolu nutný člověk, který daný test nastaví a spustí v uživatelském rozhraní nástroje BrowserStack. Až poté je možné vyhodnotit výstup testu. Vzhledem k veřejně dostupnému API pro tento nástroj by se dala celá kontrola automatizovat. Kontrolované stránky by byly zadány v XML souboru výčtem jejich URL adres a výstup kontroly by byl uložen do souboru. Stačilo by tedy, aby byl zkontrolován pouze výsledný soubor a z výsledků vyvozeny patřičné kroky.

## 4.5 Doba prvního načtení stránky

Doba prvního načtení stránky, neboli First Page Load, je důležitý atribut, který může napovědět, když něco není správně optimalizováno. Kontrola této doby se provádí pouze jednou za čas a optimalizuje se pouze v případě, že je překročena jistá hodnota. Bylo by možné napsat program, který by tuto dobu měřil a porovnával ji se zadanou referenční hodnotou. V případě, že by byla naměřená hodnota vyšší, než referenční hodnota, výstupem programu by bylo upozornění na tuto skutečnost. Díky tomu by bylo možné podchytit neoptimalizovanou část kódu například po konci každého sprintu.

## 4.6 Mezní hodnota výkonu

V modulech, pro které to dává smysl, se před vydáním nových verzí měří mezní hodnoty, díky kterým je možné ověřit, kolik kontaktů, aktivit, či jiných objektů, Kentico CMS podporuje a je schopno zpracovat. Automatizace by spočívala v napsání programu, který by v jistých časových intervalech simuloval takový test. Nebylo by tedy zapotřebí testovat dané moduly manuálně, ale pouze kontrolovat výsledky testů, zdali nebyly mezní hodnoty překročeny.

## 4.7 Pokrytí kódu v testech

Pokrytí kódu v testech je kontrolováno programem dotCover v prostředí TeamCity [1]. Testy jsou samy o sobě automatizované, ale výsledky jsou dostupné pouze v prostředí TeamCity, kde je nutné být přihlášen a proklikat se k danému testu. Možná je automatizace opět přes REST API stahovat výsledné hodnoty testů a ty uložit například do souboru v lokálním uložišti. Pověřená osoba by pak jen kontrolovala výstupní soubor a z výsledků vyvodila závěry.

# Kapitola 5

# Návrh automatizace

Na základě teoretických znalostí a závěrů provedené analýzy z předchozích kapitol je v této části práce popsán koncept navržené automatizace. Byla vybrána statická analýza všech CSS souborů v softwaru Kentico CMS pomocí nástroje Stylelint [9]. Stylelint patří do Node Package Manager (zkráceně Npm) [3], který slouží jako správce balíčků pro Javascript. Zároveň je to největší registr softwaru na světě. Tento správce balíčků je automaticky nainstalován společně s Node.js [12], což je softwarový systém navržený pro psaní vysoce škálovatelných internetových aplikací v jazyce Javascript.

Podkapitola 5.1 shrnuje požadavky pro automatizaci, které byly stanoveny. Jejich dodržení je důležité, aby bylo možné tuto automatizaci při vývoji v Kentico CMS využívat. Následně jsou v podkapitole 5.2 navrženy postupy jak a v jakém prostředí automatizaci testovat.

## 5.1 Požadavky na automatizaci

Jako první je nutné zjistit a shromáždit vhodná pravidla pro nástroj Stylelint, která mu budou předložena a v rámci DoD se hodí pro Kentico CMS. Pravidel existuje celá řada a je důležité vybrat ta, která se obecně používají nebo jsou považována za standard pro CSS soubory. Za dobu vývoje softwaru byly CSS soubory vytvářeny různými programátory, takže je více než pravděpodobné, že bude možné nalézt rozdíly mezi jednotlivými soubory i pouhým okem.

Tuto kontrolu je nutné přidat k již existujícím testům do prostředí TeamCity tak, aby se sama spouštěla a vyhodnocovala po každé změně ve sdíleném repozitáři se zdrojovým kódem Kentica CMS.

Skript musí umět sám vyhledávat všechny existujíci soubory ve struktuře programu Kentico CMS. Je to nutné proto, jelikož se CSS soubory u takto obsáhlého programu vyskytují na různých místech a bylo by značně neefektivní spouštět stejný nástroj na několika místech současně.

Cíl automatizace je unifikování všech existujících CSS souborů. To znamená, aby procházely bez chyb nástrojem Stylelint a automatizace tak byla použitelná ve skutečném procesu vývoje.

## 5.2 Testování automatizace

Statická analýza v oblasti informačních technologií vyjadřuje, že se aplikují sady metod pro analýzu programu bez nutnosti tento program spouštět. V našem případě to znamená, že se testuje obsah CSS souborů, avšak ne jejich vliv po aplikování či načtení. Díky statické analýze lze tedy garantovat jistou úroveň obsahu souborů, nikoliv jejich účinek.

Pro zjednodušení testování je vhodné prvně spustit nástroj Stylelint v konzoli operačního systému nad lokální verzí aktuálního zdrojového kódu Kentica CMS a opravit nalezené chyby. Až poté může být spuštěn nad produkční verzí ve sdíleném repozitáři.

Za účelem otestování funkčnosti bude změněn obsah CSS souborů proto, aby nevyhovoval nějakému z pravidel. Po nahrání těchto změněných souborů zpátky do repozitáře musí nástroj na tyto chyby upozornit v reportu daného testu. Všechny testy musí proběhnout bez chyby a výsledek testu musí být označen zelenou barvou se statusem *Success*.

Aby se dala automatizace považovat za funkční, je nutné nechat nástroj běžet několik dní společně s ostatními testy a sledovat minimálně jedenkrát denně, že nástroj v případě změn správně reportuje nalezené chyby.

# Kapitola 6

# Implementace a vyhodnocení automatizace

V této kapitole je popsána samotná implementace automatizace, která vychází z návrhu uvedeném v předchozí kapitole 5. Podkapitola 6.1 uvádí detaily o vývojovém prostředí, ve kterém byl nástroj použit. Dále jsou pak v podkapitolách 6.2 a 6.3 popsány způsoby použití a nastavení klíčových nástrojů. Dále je zde demonstrována funkčnost nástroje ve skutečném vývojovém prostředí Kentica.

## 6.1 Vývojové prostředí

Automatizace probíhala na stolním počítači s operačním systémem Microsoft Windows 7 nad zdrojovým kódem Kentica CMS spustitelným ve vývojovém prostředí Microsoft Visual Studio 2015 se studentskou licencí poskytovanou od VUT FIT [15].

Kromě konzole Windows PowerShell [6] byl použit server TeamCity, který slouží k průběžné integraci (angl. „Continuous integration“). Průběžná integrace pomáhá urychlit vývoj softwaru a zlepšuje spolupráci týmů.

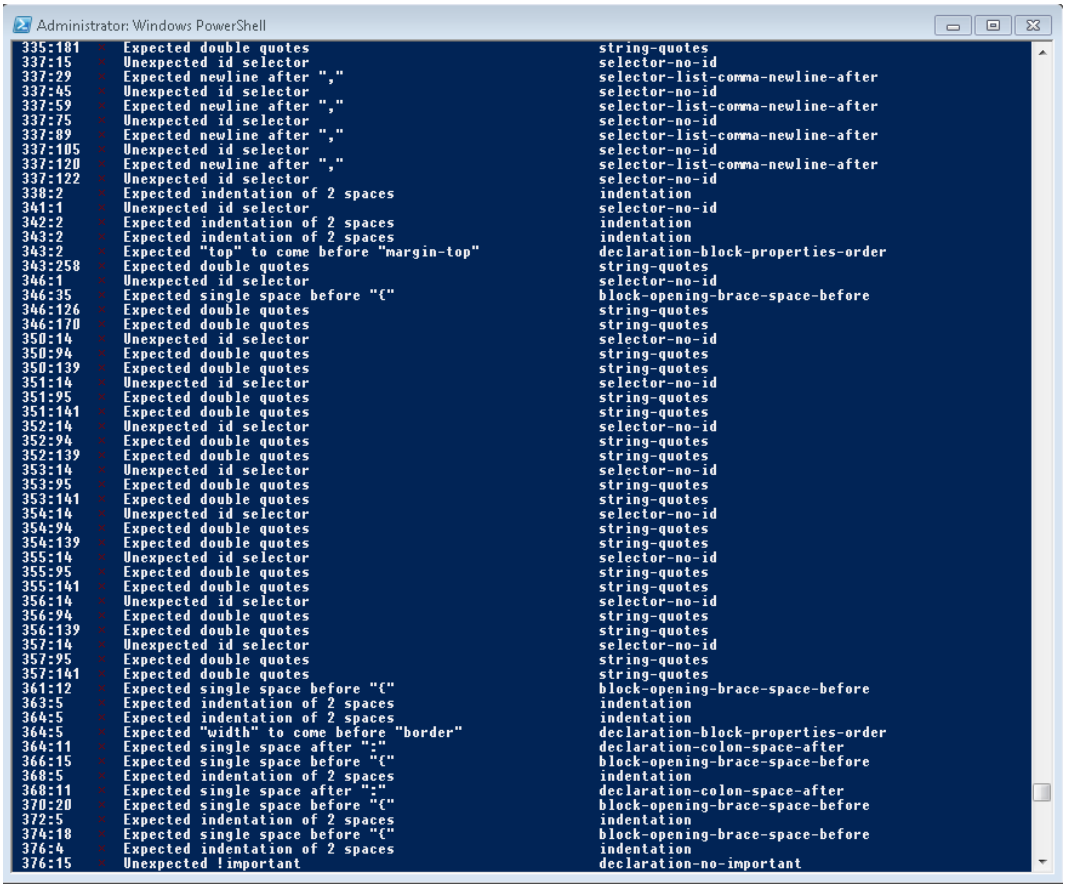
Ke správě kontrolních pravidel byl použit editor Notepad++ [16], který je dostupný volně ke stažení a nabízí výkonné funkce.

## 6.2 Stylelint

Tento nástroj musel být nejprve nainstalován příkazem `npm install -g stylelint` v kořenovém adresáři Kentica CMS. Parametr `-g` zajišťuje, že je nástroj nainstalován globálně pro všechny uživatele počítače.

Stylelint po spuštění načítá pravidla pro kontrolu ze souboru *.stylelintrc*, který byl vytvořen po instalaci v Notepad++. Kromě volně dostupného standardu nejběžnějších pravidel *stylelint-config-standard* [22] bylo přidáno ještě pár vlastních po zrevidování interních pravidel DoD týkajících se CSS stylů v Kentico CMS. Ukázku výstupu nástroje v konzoli po prvním spuštění znázorňuje obrázek 6.1.

Jako velmi užitečný se osvědčil parametr `--fix`, který umí sám opravit většinu standardních chyb přímo ve zdrojových souborech. Na výstupu po spuštění s tímto parametrem pak byly všechny neopravené chyby. Většinou se jednalo o překlepy, které musely být ručně opraveny v příslušných CSS souborech.




Obrázek 6.1: Ukázka výstupu po prvním spuštění nástroje Stylelint.

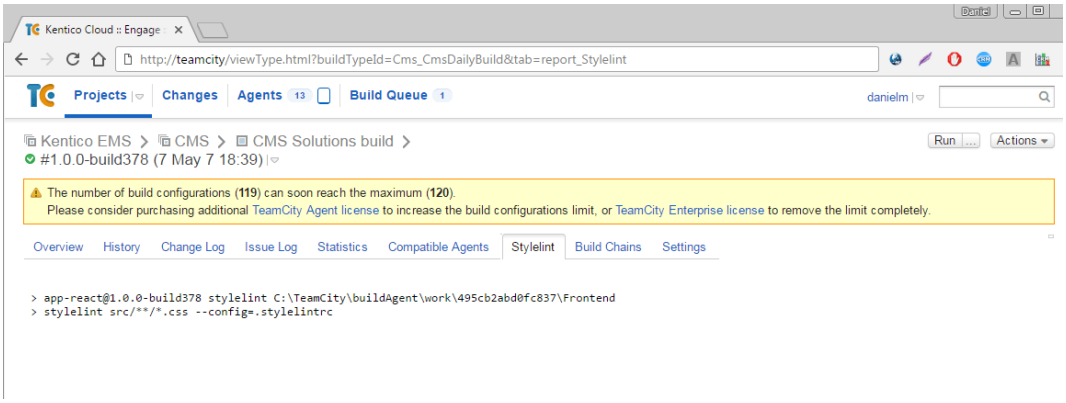
Voláním příkazu `stylelint "**/*.css"` je zajištěno, že se nástroj rekurzivně dívá do všech adresářů, podřazených adresářů a kontroluje nalezené CSS soubory, což byl jeden z požadavků.

## 6.3 TeamCity

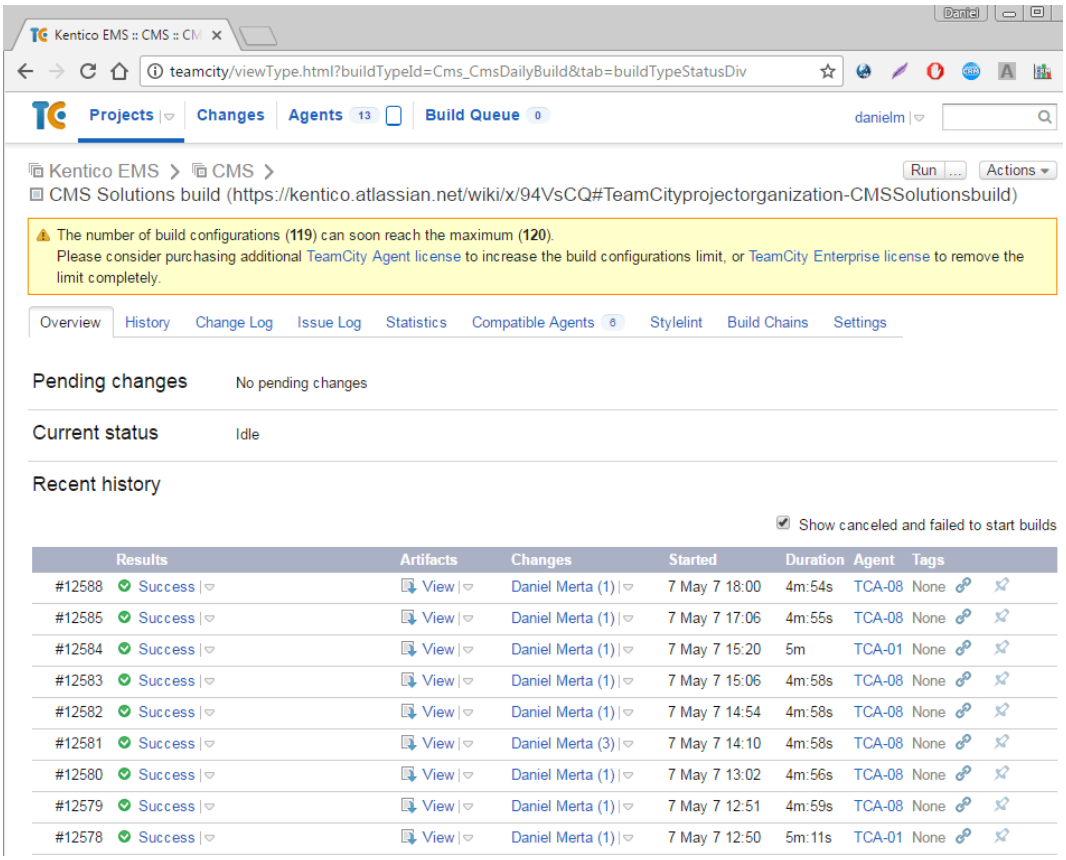
V prostředí TeamCity bylo vytvořeno sestavení o jednom kroku typu *MSBuild*, což je systém pro sestavení projektů pro Visual Studio. Tím je zaručeno, že se při každé změně ve sdíleném repozitáři projekt přeloží. Poté byl k sestavení přidán již dříve zmíněný příkaz pro spuštění nástroje Stylelint, jehož výstup lze vidět na obrázku 6.2.

## 6.4 Vyhodnocení

Jestliže se v testech nevyskytla chyba a všechny pravidla platí, výsledný report je prázdný. Pro ověření funkčnosti automatizace byl nástroj ponechán zapnutý několik hodin, během kterých došlo k devíti změnám v sestavení. Z obrázku 6.3 je patrné, že všechny testy skončily úspěšně se statusem *Success*. Lze tedy říct, že všechny CSS soubory jsou již unifikovány podle zadaných pravidel. Když dojde ke změně v některém z CSS souborů a kterékoliv pravidlo neprojde, ihned po sestavení bude zřejmé, že test nebyl úspěšný a v reportu bude označen konkrétní soubor s chybou.




Obrázek 6.2: Report nástroje Stylelint v prostředí TeamCity.




Obrázek 6.3: Úspěšně dokončené testy nad řešením Kentico CMS.

# Kapitola 7

# Závěr

Cílem této práce bylo analyzovat možné procesy k automatizaci ve společnosti Kentico Software s.r.o. a z nich vybranou automatizaci naimplementovat. Důležitou částí práce před samotnou implementací bylo nastudování jednotlivých pravidel v rámci Definition of Done a následná analýza možných automatizací. Na základě všech předchozích znalostí byla navržena automatizace statické analýzy CSS souborů v prostředí TeamCity.

Existuje více způsobů jak podobnou automatizaci naimplementovat, například pomocí API programu CSS Lint, avšak díky snažší integraci nástroje Stylelint do Teamcity byla zvolena právě tato možnost.

Ověření funkčnosti nástroje proběhlo v reálném vývojovém prostředí a z výsledku vyplývá, že se automatizace podařila a CSS soubory byly unifikovány. Navíc se nejednalo o jednorázovou akci, protože tato automatizace bude používána při vývoji Kentico CMS i v budoucnu.

Podobná automatizace jako tato, o níž je napsána bakalářská práce, by šla implementovat například pro Javascript soubory pomocí nástroje ESLint, což je obdoba Stylelint, pouze pro Javascript, nebo pro jiné typy souborů. K automatizování jiných částí Definition of Done uvedených v analýze by se dalo vycházet ze stejné myšlenky, avšak zapotřebí by byly jiné přístupy a nástroje.

# Literatura


[1] TeamCity. JetBrains, 2000, [Online; navštíveno 6. 5. 2017].
URL https://www.jetbrains.com/teamcity

[2] Designing Custom Exceptions. Microsoft, 2010, [Online; navštíveno 10. 5. 2017].
URL https://msdn.microsoft.com/en-us/library/ms229064(v=vs.100).aspx

[3] npm. Isaac Z. Schlueter, 2010, [Online; navštíveno 6. 5. 2017].
URL www.npmjs.com

[4] Agile and Related Methodologies. Gurtejpsingh, 2013, [Online; navštíveno 20. 3. 2017].
URL http://websnare.com/blog/burndown-charts-track-progress

[5] The Product Owner Role: A Stakeholder Proxy for Agile Teams. Agile Modeling, 2014, [Online; navštíveno 12. 5. 2017].
URL http://agilemodeling.com/essays/productOwner.htm

[6] Windows PowerShell. blogCZSK, January 2014, [Online; navštíveno 15. 5. 2017].
URL https://blogs.technet.microsoft.com/technetczsk/2014/01/08/seril-windows-powershell-vylepen-konzole-st-42/

[7] Annotated ECMAScript 5.1. ECMAScript, 2015, [Online; navštíveno 30. 4. 2017].
URL http://es5.github.io/#x7.6.1

[8] Burndown chart. WEBSNARE, 2015, [Online; navštíveno 15. 5. 2017].
URL http://websnare.com/blog/burndown-charts-track-progress

[9] Stylelint. npm, October 2016, [Online; navštíveno 23. 4. 2017].
URL https://stylelint.io

[10] JavaScript HTML DOM. w3schools, 2017, [Online; navštíveno 10. 5. 2017].
URL https://www.w3schools.com/js/js_htmldom.asp

[11] JavaScript Use Strict. w3schools, 2017, [Online; navštíveno 5. 5. 2017].
URL http://www.w3schools.com/js/js_strict.asp

[12] Node.js. 2017, [Online; navštíveno 23. . 2017].
URL https://nodejs.org

[13] Product Owner. Mountain Goat Software, 2017, [Online; navštíveno 4. 5. 2017].
URL
https://www.mountaingoatsoftware.com/agile/scrum/roles/product-owner

[14] Using Standard Exception Types. Microsoft, 2017, [Online; navštíveno 20. 3. 2017].
URL https://msdn.microsoft.com/en-us/library/ms229007(v=vs.110).aspx

[15] Visual Studio. Microsoft, 2017, [Online; navštíveno 1. 5. 2017].
URL https://www.visualstudio.com

[16] Notepad++. Don Ho, 2016, [Online; navštíveno 7. 5. 2017].
URL https://notepad-plus-plus.org/

[17] Henrik Kniberg, M. S.: *Kanban and Scrum: making the Most of Both*. Lulu.com, 2010, ISBN 978-0-557-13832-6, [cit. 2017-04-21]. Dostupné z: https://www.infoq.com/minibooks/kanban-scrum-minibook.

[18] Keith, C.: *Agile game development with Scrum*. Upper Saddle River, 2010, ISBN 978/032/1618/528, 340 s.

[19] Šlapák, J.: Aplikace pro sledování a vykazování výkonů. 2012.

[20] Markham, D.: *ScrumMaster*. a tiny giant book, 2012, [cit. 2017-05-01]. Dostupné z: http://tiny-giant-books.com/scrummaster.htm.

[21] Myslín, J.: *Scrum, Průvodce agilním vývojem softwaru*. Computer Press, 2016, ISBN 9788025146507.

[22] Stylelint-config-standard. npm, [Online; navštíveno 28. 4. 2017].
URL https://github.com/stylelint/stylelint-config-standard

# Příloha A

# Obsah přiloženého paměťového média

- **merta_bakalarska_prace** - kompletní elektronická podoba
- **.stylelintrc** - soubor se vstupními pravidly pro nástroj Stylelint
- **stylelint_report_messages.txt** - ukázkový výpis zpráv nástroje Stylelint při nalezení chyb
- **projekt.pdf** - technická zpráva ve formátu PDF