FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Gamifikace testování software
Student:	Bc. Pavel Čejka
Vedoucí:	Ing. Petra Pavlíčková, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

Cílem diplomové práce je prozkoumat implementační možnosti gamifikačních prvků v testovací fázi vývoje software.

1. Prozkoumejte a nastudujte odbornou literaturu k problematice gamifikace v rámci vývoje software.
2. Identifikujte příležitosti pro implementaci gamifikačních prvků v testovací fázi.
3. Navrhněte realizaci jednotlivých gamifikačních prvků.
4. Implementujte prototyp realizující některé z identifikovaných a navržených prvků a jeho otestování.
5. Zhodnoťte navržené řešení (jak z pohledu technologického, tak z pohledu ekonomického) a doporučte další postup.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 11. ledna 2018

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Diplomová práce

Gamifikace testování software

Bc. Pavel Čejka

Vedoucí práce: Ing. Petra Pavlíčková, Ph.D.

5. května 2018

Poděkování

Chtěl bych poděkovat vedoucí své práce, doktorce Pavlíčkové, za podnětné diskuze a odborné vedení. Dále bych chtěl poděkovat své rodině za skvělé zázemí a morální podporu, bez které by tato práce nemohla být dokončena.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 5. května 2018 .....

Odkaz na tuto práci

Čejka, Pavel. *Gamifikace testování software.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Tato diplomová práce se zabývá možnostmi aplikace gamifikace, jakožto populárního trendu současné doby, na procesy testování software. Cílem práce je prozkoumat možnosti gamifikace, které se pro tuto oblast nabízejí. Nejprve se v práci připomíná důležitá teorie ohledně nutnosti testování a několika druhů testování, poté je vysvětlena gamifikace, důvody, proč gamifikovat a způsoby, jakými lze gamifikovat. Práce se pak zaměřuje na možnosti přidání gamifikačních prvků do programu JMeter a jemu příbuzných programů. V praktické části diplomové práce jsou analyzovány nedostatky programu JMeter a navržena architektura nového vizualizéru aplikující vybrané gamifikační prvky. Přestože vznikl demonstrativní prototyp, výstupem práce je především několik doporučení a směrů, kterými se vývoj gamifikace může nadále ubírat.

Klíčová slova gamifikace, JMeter, testování, vizualizace

Abstract

This thesis explores the possibilites of application of gamification - a popular trend of today - into the processes of software testing. The aim of the thesis is to assess which of these possibilities are viable for this field. The thesis deals

initially in the necessary theory about the need of testing and about different types of testing, then gamification is introduced alongside reasons why and how is gamification achievable. In the practical part of the thesis, the drawbacks of JMeter software are analyzed and new visualizer was designed which uses selected gamification principes. Despite the proof of concept being created, main outputs of the thesis are several suggestions and paths for gamification to move along to the future.

Keywords gamification, JMeter, testing, visualization

Obsah

Seznam obrázků

Úvod

Gamifikace je moderní trend, který postupně proniká do více a více oblastí obchodního života. O gamifikaci čehokoli mluvíme, když se do dané oblasti začnou vkládat prvky přebírané z (nejen počítačových) her. Typickým cílem gamifikace je angažovat cílovou osobu a tím zpříjemnit vykonávání rutinní nebo nudné části procesu.

V současnosti je gamifikace spojena především s prvky marketingu, ale postupně se dostává i do interních procesů. Tato práce se zaměří na prozkoumání možností průniku gamifikačních prvků do procesů tvorby software, konkrétně pak do procesů v průběhu fáze testování vyvíjeného software.

V úvodní části dojde ke krátké rekapitulaci důležitých termínů a klasifikace, které jsou používany ve spojení se softwarovým testováním. Bude vysvětleno, proč je nutné testovat a z jakých částí se testování skládá. Poté budou prozkoumány možnosti gamifikace vhodných částí testování. V závěru bude jedna z těchto gamifikačních možností vybrána a demonstrována pomocí proof of concept.

Při psaní této práce a při vývoji prototypu demonstrujícího zvolený gamifikační princip vycházím nejen z citovaných zdrojů, ale zejména ze své praxe a zkušeností programátora a názorů spolupracovníků.

Kapitola 1

Testování software

Testování software je nedílnou a velmi důležitou součástí softwarového vývoje. Různé formy testování probíhají v průběhu celého procesu vývoje a jak pokračuje vývoj software, mění se i metodika a principy testování a ověřování funkčnosti. Publikace Foundations of Software Testing[1] vymezuje pět základních fází vývoje z hlediska funkcionality, a to

1. **Requirements** - Specifikace požadavků. V této fázi probíhá ujasnění s klientem, jaké funkcionality má program obsahovat. Odpovídá na otázku „Co má software dělat?“

2. **Design** - Návrh aplikace. V této fázi se plánuje, jaké principy bude software používat k dosažení definovaných funkcionalit. Odpovídá na otázku „Jak to má dělat?“

3. **Build** - Vystavění aplikace. Programátorská fáze. Realizace požadavků dle navrženého designu.

4. **Test** - Testování aplikace. Zde je důležité si uvědomit, že testování softwaru, které je předmětem této diplomové práce, probíhá průběžně. Tato fáze se zaměřuje spíše na testování funkcionalit, zda jsou splněny požadavky.

5. **Live use** - Také Delivery. Jedná se o fázi podpory, kdy software je už vyvynutý, předaný zákazníkovi. V této chvíli jsou programátorské práce už především reaktivní, reflektující změny v businessu nebo legislativě, nebo případné opravy bugů, které prošly filtry quality control procesů.

Chyby mohou nastat (a typicky nastávají) ve všech pěti fázích vývoje. Tyto chyby se pak přenášejí i do dalších fází.

Requirements		Správně formulovaný požadavek		Správně formulovaný požadavek		Správně formulovaný požadavek		Správně formulovaný požadavek		Špatně formulovaný požadavek
Design		Správně navržené řešení požadavku		Správně navržené řešení požadavku		Správně navržené řešení požadavku		Špatně navržené řešení požadavku		Správně navržené řešení požadavku
Build		Správně naprogramováno		Správně naprogramováno		Špatně naprogramováno		Správně naprogramováno		Správně naprogramováno
Test		Produkt funguje, jak se očekávalo		Nevhodně otestováno, produkt vykazuje bugy, i když je neobsahuje		Produkt obsahuje bugy		Produkt neodpovídá zadání		Produkt odpovídá zadání, ale neodpovídá představám
Deployment		Může být předáno k nasazení		Může být předáno k nasazení po vyjasnění problémů s testy.		Defekty jsou lehce opravitelné a lze nasadit s mírným zpožděním		Produkt je špatně navržen. Nelze odevzdat, je třeba přepracovat		Tyto chyby nemusí IT tým vůbec odhalit, protože vycházejí z již špatného zadání

Obrázek 1.1: Tabulka propagace chyb fázemi vývoje. Inspirováno Foundations of Software Testing[1]

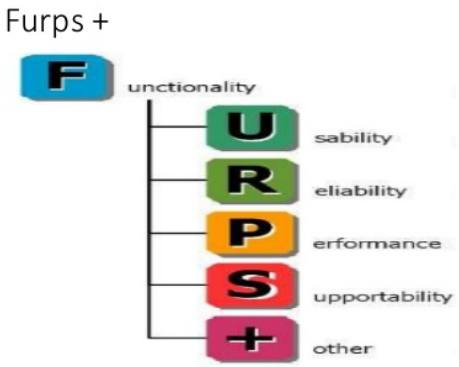
Obr. 1.1 ukazuje, jak se chyby propagují z fáze vývoje, ve které vznikla, do dalších. Je prakticky nemyslitelné, že by se chyba „sama opravila“ v některé z dalších fází, protože ani sebekvalitnější programátor nedokáže vykrýt špatně navrženou nebo dokonce špatně zadanou funkcionalitu. V červené fázi nastala chyba, žluté fáze jsou pak touto chybou ovlivněny a musí být přepracovány po odstranění původní chyby.

1.1 Druhy testování

Existuje větší množství druhů testů. Tyto se dělí podle množství kritérií. V této kapitole se zaměřím především na *míru uživatelského zásahu* a *důvod testování*. Samozřejmě existují i další kritéria, podle kterých lze testy dělit, nicméně tyto dvě poskytují znalostní základ, na který se budu odvolávat při gamifikaci.

1.1.1 Důvod testování

Každý test má své opodstatnění a svůj důvod. Laikové si při slově „testování“ vybaví především funkční testy. Tedy testy, které odpoví na otázku „Dělá to to, co má?“. Tyto testy jsou důležitou, nikoli však jedinou oblastí testování. Robert Grady a Deborah Caswell stanovili metodiku FURPS[2]. Přestože se jedná o téměř třicet let starou publikaci, rozdělení je nadčasové a zachovává si svou platnost i do dnešních dní.



Obrázek 1.2: Vizualizace oblastí FURPS

Metodika FURPS zmiňuje následující oblasti (obr. 1.2[3]):

1. **F-functionality**: funkční testy. Ověřují, zda jsou splněny funkční požadavky

2. **U-usability**: Ověřuje aplikaci jako celek. Typicky se provádí na straně zákazníka a neslouží k zhodnocení funkčnosti, jako spíše k zhodnocení lidských faktorů, tedy jak jednoduché je s programem pracovat, estetika, srozumitelnost a konzistence GUI. Rovněž se mohou zaměřovat nejen na hlavní delivery, tedy software, ale i na podpůrné delivery, jako je například dokumentace, výukové materiály, průvodci nastavením a podobně.

3. **R-reliability**: Testují robustnost programu na metrikách, jako jsou například citlivost aplikace na pády, rychlost a obtížnost obnovy nebo průměrný čas mezi selháním.

4. **P-performance**: Ověřuje výkon aplikace. Rychlost běhu, rychlost odezvy, přesnost, dostupnost a efektivita využití přidělených zdrojů, to všechno jsou důležité metriky pro performance testy.

5. **S-supportability**: Testují míru kompatibility programu s ohledem na rozšiřitelnost, dostupnost rozhraní, konfigurovatelnost a další.

Doplňkové popisy FURPS vycházejí z prezentace Adastra Apliqua[4].

1.1.2 Míra uživatelského zásahu

Tato část se věnuje rozdělení testů podle míry uživatelského zásahu. Tím se myslí, na kolik je vyžadována spoluučást testera na provedení testu. Toto rozdělení je vhodné provést do následujících tří kategorií:

1. Manuální testování
2. Semi-automatické testování
3. Plně automatické testování

1.1.2.1 Manuální testování

Jedná se o situace, kdy tester činí všechny kroky testu osobně. Typicky tester dostane od test managera seznam test case, které je třeba ověřit a postup, kterým je možné tohoto ověření dosáhnout. Při tomto druhu testování je většinou tester posazen do role uživatele systému a ověřuje funkčnost systému právě z pohledu uživatele. Z pracovních zkušeností vyvozuji, že testeři v této roli jsou zodpovědni za nalezení chyb a způsobu, jak je replikovat a za co nejkvalitnější nahlášení chyby svému test managerovi nebo do Quality Center programu. Nejsou ovšem zodpovědni za hledání příčiny chyby.

Manuální testování nemá smysl na malých funkčních jednotkách software, spíše se používá pro kompletní otestování software před dodáním, a to zejména na straně zákazníka při akceptačních testech.

1.1.2.2 Semi-automatické testování

Tento druh testování se opírá o kombinaci testovacího software a rozhodování testera. Oproti manuálnímu testování jsou jednotlivé kroky test case automatizované a tester pouze vybírá, jaké test case chce spustit. Testovací software se postará o nezbytné provedení kroků k dokončení tohoto test case a testerovi sdělí výsledek.

Semi-automatické testování slouží k otestování menších funkčních celků a tradičně jde více do hloubky. Například se může testovat, jaké zprávy si posílají jednotlivé moduly software nebo jak fungují dílčí operace větší funkcionality. Tato úroveň detailu bývá od výše popsaného manuálního testování odstíněna.

Testeři zde už mají přístup k více detailům funkcionality a mohou být pověřeni i nalezením příčiny zjištěné chyby.

1.1.2.3 Plně automatické testování

Plně automatické testování bývá většinou v gesci integračního software, jako je například Jenkins. V tomto případě se testují spíše malé jednotky kódu. Funkcionalita SQL příkazů, funkční build a podobně. Tyto testy bývají tak specifické, že odhalení chyby rovnou ukáže i na její příčinu.

Druhou variantou, jak přistoupit k automatickému testování, je rozšířit semi-automatické testování o automatizační skript, který sám bude zapínat nastavené test case v nastavených hodinách. Výhodou tohoto přístupu je možnost odsunout spuštění časově náročných testů mimo pracovní dobu, takže tester přijde další den již k dokončeným testům a může z výsledků vyvozovat

závěry. Nicméně ani tento přístup se neobejde bez vstupu testera do testovacího procesu, protože výsledky vzniklé automatickým testováním se musí interpretovat.

Kapitola 2

Gamifikace

2.1 Motivace pro gamifikaci

Gamifikace je poměrně nový termín, pod kterým se schovává množství postupů a metod, které mají za úkol přitáhnout a udržet pozornost pomocí různých herních prvků v neherních oblastech. Gamifikace se v současné době využívá zejména jako marketingový nástroj. Může mít podobu virtuálních odznaků, bodů v aplikaci, ale i spousty dalších. Při gamifikaci se meze nekladou. Typickým příkladem gamifikované aplikace je Foursquare[5]. Tato geolokační služba nabízí propracovaný systém bodování na základě navštěvování klíčových bodů. Vzhledem k pravidlům se pak mohou svádět boje mezi uživateli o nejvíce bodů spojené s návštěvou daného místa. To otevírá nové marketingové možnosti.

Po úspěchu gamifikace v marketingu se začaly gamifikační principy hlásit o slovo i v dalších odvětvích, přičemž nejvýraznější z nich jsou různé druhy managementu. Management je o vytváření důležitých rozhodnutí a důsledných reportech. Obzvláště tvorba reportů a sbírání feedbacku od zaměstnanců může být velmi zdlouhavá a únavná pro obě strany, vznikly tedy interaktivní formuláře s přívětivou grafikou, která upoutá pozornost cílového člověka a z vyplňování reportů udělá úkol přinejmenším snesitelný, v lepším případě zábavný. Tím ale gamifikace v pracovním prostředí nekončí. Petr Kočí ve svém článku[5] sděluje: „Před dvěma lety Byron Reeves a J. Leighton Read v knize Total Engagement: Using Games and Virtual Worlds to Change the Way People Work and Businesses Compete předpověděli, že týmová spolupráce, motivační systém a další prvky známé z počítačových her brzy od základu změní svět práce a podnikání. Hranice mezi povinnostmi a zábavou se podle nich brzy rozmaže. Podle Jesse Schella, zkušeného herního designera a učitele herních studií na Carnegie-Mellonově univerzitě se rozostří hranice mezi hrou a téměř vším ostatním.“, můžeme tedy brzy čekat další gamifikační ofenzívu. Principy a cíle gamifikace vizuálně ukazuje obr. 2.1[6]

Gamifikace mimo marketing má ale i své nevýhody. Většina gamifikace

Obrázek 2.1: Vizualizace fungování gamifikace

sebou nese inherentní zhoršení maximálního výkonu. Uvedu na příkladu, kdy animovaní skřítci přitáhnou formulář k vyplnění a nadále kolem něj skákají a ukazují, která políčka je třeba ještě vyplnit. Toto je dobrý příklad gamifikace. Nicméně takový formulář nikdy nepůjde vyplnit tak rychle, jako by šel vyplnit strohý formulář plně motivovaným zaměstnancem. Vyplňování formulářů ale nebývá záležitost, kterou lidé dělají na plno. Tedy pokud mám potenciál 100%, ale realitu 50%, gamifikace může snížit potenciál např. na 90%, ale realita se zvýší na 70%, tedy ve výsledku vidíme zlepšení. Hlavní myšlenka tohoto odstavce je ta, že zatímco gamifikace reklamních a marketingových kampaní se prakticky nedá „přepálit“ a celý model lze založit na hře, využívání gamifikace jako podpůrného prostředku pro pracovní účely je zapotřebí využívat v rozumné míře a jen na vhodných místech.

Autoři Werbach a Hunter ve své knize For the Win: How Game Thinking Can Revolutionize Your Business[7] potvrzují, že jedním z klíčových cílů gamifikace v businessu je motivace. Motivací jsem se v jiném kontextu zabýval ve své bakalářské práci[8], kde jsem rozdělil druhy motivace na **vnitřní** a **vnější**, přičemž gamifikace může působit na oba druhy motivace v závislosti na její implementaci.

1. **Vnitřní motivace** - stav mysli jedince, který má své vlastní důvody pro to být motivován. Tyto důvody mohou být vyvolány například tím, že člověk spatřuje přínos v práci, kterou vykonává, nebo v ní vidí příležitost k sebezdokonalení. Další z důvodů vnitřní motivace je samozřejmě to, že

konkrétní činnost člověka baví. Na tento poslední důvod cílí gamifikace zaměřená na vnitřní motivaci.

2. **Vnější motivace** - je vyvolána vnějšími stimuly, obvykle, i když ne nezbytně, nadřízenou osobou. Vnější motivace může být jak pozitivní - pochvala, prémie, kariérní růst, i negativní - různé sankce. Gamifikace vydávající se cestou vnější motivace bude nejčastěji zaměřena na systém benefitů výměnou za získané body, tedy pozitivní motivaci.

2.2 Gamifikační prvky

Jako gamifikační prvky (též herní prvky) se označují metody převzaté z her. Nejjednodušším gamifikačním prvkem, který se často používá, je tabulka hodnocení (leaderboard), podobně jako bývala na konci starých arcade her. Samozřejmě, že gamifikačních prvků existuje celá řada. Kniha For the Win[7] rozděluje gamifikační prvky do tří skupin:

- Zákonitosti
- Mechaniky (též Herní mechaniky)
- Komponenty

Tyto skupiny pak dále autoři popisují ve své druhé knize, The Gamification Toolkit: Dynamics, Mechanics, and Components for the Win[9].

2.2.1 Zákonitosti

Jako zákonitosti se označují prvky, které utvářejí rámec pro gamifikovaný systém. Mezi zákonitosti se například řadí různá omezení, vyprávění (nějakého příběhu) nebo pravidla pro postup ve hře. Obecně zákonitosti tvoří možnosti hry. Nastavené zákonitosti určí žánr hry, tempo, počet hráčů a podobné atributy, ale nejsou přímo součástí aktivní hry.

Zákonitosti pro hru plní podobnou úlohu jako architektura pro tradiční software.

2.2.2 Mechaniky

Herní mechaniky jsou to, co zapojuje hráče do hry. Werbach a Hunter identifikovali[9] deset základních druhů mechanik, a to tyto:

- Výzvy
- Náhody
- Spolupráce

- Soupeření
- Zpětná vazba
- Získávání zdrojů
- Odměny
- Transakce
- Kola - ve smyslu *rozdělení na to, kdo zrovna hraje*
- Výherní podmínky

Na první pohled nesourodý seznam dává smysl, když se na něj nahlédne z jiného úhlu pohledu. Výše uvedený seznam pokrývá téměř všechny oblasti, které hru dělají hrou. Čili spíše než pohled *Co z tohoto zapracujeme* je lepší použít optiku *Jak zapracujeme každý z těchto bodů*, přičemž i odpověď „nijak" je přípustná. Jednodušší hry využívají pouze podmnožinu těchto oblastí. Například *šachy* ve svém jádru implementují pouze *soupeření*, *kola* a *výherní podmínky*. Existují ale i další varianty šachů, které implementují prvky *spolupráce*, na turnajích se přidávají prvky *odměn*.

Herní mechaniky tedy popisují, jak se hra hraje. Spolu se *zákonitostmi* tvoří pravidla hry a identifikují hru. Pokud dva programátorské týmy dostanou seznam zákonitostí a herních mechanik (ale žádné komponenty, viz. 2.2.3), měly by ve výsledku doručit příbuzné, nicméně ne identické hry.

2.2.3 Komponenty

Komponenty jsou konkrétní implementací herních mechanik a zákonitostí. Jedná se o nejnižší úroveň návrhu hry před samotnou implementací. Komponenty řeší takové věci, jako je vzhled mapy/hrací plochy, vzhled avatarů nebo figur, záchytné body, důležité události, souboje s bossy, herní úrovně a podobné detaily, které dotvoří hru v jedinečné dílo.

Komponenty už typicky nemají vliv na druh hry nebo příliš abstraktní zadání. Vliv komponent uvedu na příkladu srovnání dvou her z 90. let minulého století.

Wolfenstein 3D proti Doom. Obě tyto hry pocházejí od Johna Romera, potažmo Id Software a mají téměř identické zákonitosti a herní mechaniky. Obě hry jsou first-person střílečky v 3D prostoru, obě jsou rozdělené do kapitol, které jsou dále dělené na úrovně. Obě mají totožné podmínky výhry (dojít na konec úrovně, nebo zabít bosse) i prohry (klesnutí životů hráče na nulu). Podobností by se zajisté dalo najít ještě více. Nicméně obě hry se liší v téměř všech svých komponentech. Úrovně jsou naprosto odlišné, v každé hře je k dispozici jiná sada zbraní. Design nepřátel je také odlišný, ve Wolfensteinu hráč bojuje proti nacistům, v Doomu proti démonům z pekla.

Obrázek 2.2: Screenshoty z her Wolfenstein 3D a Doom

Hráči si pravděpodobně velmi rychle všimnou podobnosti jednotlivých her. Lidé se zkušenostmi z Wolfensteina se rychle adaptují v Doomu a obráceně. Nicméně rozdílné komponenty jsou to, co tyto hry odlišuje a hráči jedné hry nemají v druhé pocit „tohle tu už bylo“.

Pro referenci viz. 2.2[10][11]

2.3 Gamifikování systému

V předchozí sekci byly představeny gamifikační prvky na příkladu opravdových her. Tedy děl, softwarových či fyzických, které cílí na hráče s jediným účelem - pobavit. To není případ prakticky žádné gamifikace, ať se již gamifikuje cokoli. Gamifikovaný systém si stále musí zachovat původní funkce a účel, gamifikace pouze přidává další vrstvu systému s definovaným účelem (viz. začátek této kapitoly). Stejně tak není cílená na hráče, ale na zaměstnance. Oba tyto faktory je zapotřebí vzít v potaz při gamifikování systému.

Většina zaměstnanců by nejspíš nedocenila pravidelné pop-up se sudoku,které bylo přidáno gamifikací systému, aby udrželo pozornost. Dokonce i těm zaměstnancům, kteří sudoku luští rádi a užívali by si tyto pop-upy, by klesala výkonnost a efektivita prováděné práce.

Tím by trpěla primární funkce původního systému. Taková gamifikace není provedena dobře a škodí prakticky všem oblastem práce.

Pro to, aby se takovéto situace nestávaly, je zapotřebí dobré analýzy systému a prostředí. Kniha For the Win nabízí framework pro tuto analýzu a označuje ho jako *framework šesti D*[7]. V angličtině všechny kroky začínají na písmeno D. V českém překladu by se mohlo jednat spíše o *framework šesti P* a našroubovat překlad pomocí slov jako *popište*, *poznejte* a podobně, nicméně překlad neomezující se na jedno písmeno dokáže popsat jednotlivé kroky lépe:

1. Identifikujte cíle businessu
2. Popište žádané vzory chování
3. Poznejte vaše hráče
4. Navrhněte iteraci aktivit
5. Nezapomeňte na zábavu
6. Nasaďte příslušné nástroje

2.3.1 Identifikace cílů businessu

Zde je důležité si položit dvě základní otázky - *Proč gamifikovat?* a *Co gamifikovat?*. Z nich pak vyplývá třetí, *Čeho chceme dosáhnout?*. V základu gamifikujeme kvůli zvýšení efektivity, a to buďto přímo, dobrým navržením hry, nebo nepřímo, zvýšením motivace. Pokud manažer nedokáže odpovědět

na kteroukoli z těchto otázek, gamifikační proces nejspíš skončí podobně, jako výše uvedený příklad se sudoku.

2.3.2 Žádané vzory chování

Zde je třeba vymezit, co chceme, aby hráči dělali (a případně jak jejich výkony měřit). Všechno, co vymyslíme, by mělo nějakým způsobem směřovat k dříve vytyčeným cílům businessu. Poté, co se určí, co se bude dělat, stanoví se cíle, protože správně nvržená hra musí jít vyhrát.

2.3.3 Poznejte vaše hráče

Gamifikace se tvoří pro uživatele systému. Tito lidé již prošli významnou částí životů a jejich povahy jsou již definované a naprosto odlišné. Přestože se dají povahy diskretizovat a označit, je stále nutné počítat s rozdíly mezi jednotlivými povahami. Při vývoji gamifikovaného systému je nutné využít znalostí o cílových hráčích a jim daný systém přizpůsobit.

Bohužel, narozdíl od tradičních her, cílový uživatel gamifikovaného systému nemá typicky možnost si vybrat, že hrát nebude. Je tedy zapotřebí se vyvarovat kontroverzním motivům, jako je např. použití podobizny proroka Mohammeda v internetové hře Faith Fighter (http://www.superhry.cz/games/926/), viz. obr. 2.3. I pokud se tvůrce gamifikovaného systému úspěšně vyhne všem sociálním nástrahám, je stále nutné vzít v potaz povahu uživatelů a křivku obtížnosti, aby hráči netrávili hodiny nad hádankou místo nad pracovními úkoly, nebo naopak aby hra nebyla „piece of cake“, neboli nezajímavou rutinou kvůli tomu, že zájmy uživatelů směřují jiným směrem.

Obrázek 2.3: Vyobrazení podoby proroka Mohammeda ve hře Faith Fighter.

2.3.4 Iterace aktivit

Hra nikdy není jen jedna aktivita, jinak by životnost takové hry byla velmi nízká a neplnila by svůj účel. Hry tedy postupně iterují samy na sebe, čímž motivují hráče k zapojení a pokračování ve hře.

Nejběžnějším druhem iterace ve hrách je systém úrovní. Ať se jedná o dříve zmíněný Doom nebo třeba mobilní Candy Crush, princip je stejný. Hráč začíná na nejnižší úrovni. Stanovené podmínky k vítězství jsou poměrně jednoduché a hra neklade hráči příliš velký odpor, ať je to ve formě slabých monster nebo štědrého počtu dostupných tahů. S přibývajícími úrovněmi ale hra „přitvrzuje“ a dokončení dalších úrovní je čím dál tím obtížnější.

Druhým způsobem iterace je systém zpětné odezvy a odměn. Tento princip je typický pro sběratelské nebo adventurní hry. Tyto hry typicky nezvyšují svou obtížnost s časem, místo toho využívají lidské povahy, aby si udržely publikum.

Ve sběratelských hrách se prodlužuje čas pro získání „té poslední věci do sbírky“ aspektem náhody - tyto hry většinou fungují otevíráním balíčků s náhodným obsahem. Takže čím víc věcí ve sbírce, tím více duplikátů které sbírku nerozšíří. Většina těchto her zároveň umožňuje sbírku nějakým způsobem využívat pro další hru (např. Hearthstone, kde se sbírají karty, ze kterých hráči tvoří balíky, se kterými proti sobě hrají).

Adventurní hry pak zase po dokončení úkolu odmění hráče pokračováním příběhu. Úkoly nemusí stupňovat obtížnost, hráč je (měl by být) vtažen do hry samotným dějem, a kvůli tomu pokračuje ve hraní.

2.3.5 Nezapomenout na zábavu

Tento bod připomíná, že hra je dobrá jen pokud je zábava ji hrát.

Systém by nebyl gamifikován správně, pokud by se hráči účastnili z povinnosti a i přes nezájem o hru samotnou.

Příkladem tohoto problému může být navázání výročních odměn na určitý počet získaných bodů. Tím se zájem o hru vynucuje a gamifikace selhává ve svém základním poslání - učinit činnost příjemnější

Kapitola 3

Gamifikace vývoje software

Z kapitoly 1 plyne, že testování software je náročná práce s širokým záběrem, vyžadující kromě dobré znalosti systému také pozornost a smysl pro detail. V těchto ohledech se příliš neliší od adventurních her, nabízelo se tedy převzít některé prvky právě z her tohoto ražení. Z části 2.3 zase víme, že je zapotřebí uvážit, které aspekty testování je vůbec vhodné gamifikovat.

Gamifikace testování je prozatím zelená louka. Neexistují ucelená pojednání o gamifikaci testování ani se mi nepodařilo dohledat žádný startup, který by se zajímal problematikou vývoje gamifikovaných testovacích systémů. Při hledání jsem ovšem zjistil, že existují gamifikační utility pro task management, které by se daly použít i při testování software. Jednou z takových utilit je GetBadges (https://getbadges.io/), který funguje na principu odměn za splněné úkoly zadané v některém z kompatibilních task management programů, jako je např. Jira.

Přestože program jako je GetBadges je možné používat i při testování, jeho hlavní zaměření je na budování počítačových systémů.

3.1 Gamifikace vývoje

Martin Koliba se ve své bakalářské práci zabývá možnostmi implementace gamifikačních prvků do agilního vývoje software metodikou scrum[12]. V následující části lehce shrnu jeho závěry, které poskytnou kontext pro další kapitoly této práce.

3.1.1 SCRUM

Scrum je založený na iterativních a inkrementálních principech vývoje. Je obecně známo, že agilní metody vývoje, mezi které patří i scrum, jsou lépe schopny se adaptovat měnícím se požadavkům klienta.

Scrum identifikuje tři role účastníků:

1. **Vlastník** - Zastupuje zájmy zákazníka. Stanovuje priority a cíle a předává informace mezi Scrum týmem a zákaznickou strukturou. Přestože v praxi je následující podmínka špatně realizovatelná, *Vlastník* by měl být jediný člověk ze strany zákazníka, který se scrum týmem komunikuje.

2. **Scrum master** - Role scrum mastera je zařídit, aby scrum probíhal co možná nejplynuleji, bez vnějších vlivů. Kontroluje využívání procesů a dodržování plánů a v případě potřeby odstraňuje překážky a řídí komunikaci s vnějším prostředím. Scrum master nicméně není typický team leader. Nepodílí se na rozhodování o technické stránce vývoje a typicky není nadřízený ostatním členům týmu.

3. **Člen scrum týmu** - Scrum tým se skládá z jednotek lidí, kteří dohromady mají všechny potřebné dovednosti k vývoji software (analýza, vývoj, *testy*, atd.). Více než desetičlenný tým je těžké scrumem uřídit. Jednotliví členové plní stanovené vývojové úkoly tak, aby na konci sprintu mohli dodat tzv. *inkrement*[1].

Při vývoji podle scrumu se využívá série krátkých fází vývoje (tzv. **sprintů**). Tyto sprinty jsou časově ohraničené a funkčně oddělené.

Sprint trvá obvykle mezi 2 a 4 týdny. Během těchto týdnů probíhá každodenní mini-sprint a na konci sprintu se vytvoří zpráva o sprintu a předá se *vlastníkovi* výsledek činnosti i s touto zprávou. V ideálním případě na konci sprintu je k dispozici inkrement systému.

Sprinty a i scrum jako takový, jsou velmi citlivé na pečlivé plánování a soustředění. Nezanedbatelné množství času se stráví na rozplánování úkolů a na začátku každého dne se určí cíle na ten den. Pro scrum více než kde jinde také platí pravidlo „Dokud není něco hotovo, nezačínáme cokoli dalšího“. Sprinty jsou velmi soustředěné na jednu konkrétní funkcionalitu, kterou tým vyvine a dodá v rámci jednoho sprintu, čímž tvoří inkrementy.

3.1.2 Gamifikace scrumu

Z popisu scrumu je vidět poměrně výrazný potenciál pro gamifikaci. Scrum je založený na spoustě zalogovaných a řízených úkolů, využívání tohoto logu k určení cílů pro hru se přímo nabízí. Toho také využil Martin Koliba a ve své práci navrhl gamifikační systém založený na integraci do nástroje JIRA[12].

Zjednodušeně: Za plnění úkolů získávají uživatelé zkušenosti, které mohou směnit za právo vybarvit určitou plochu na kanvasu (klidně i přepsat kanvas spolupracovníka). Když je canvas plný, obrázek je uložen a začíná se na novo.

[1] Malá jednotka systému, která se dá nasadit k už stávajícímu řešení

3.1.3 Testování ve scrumu

Testování ve scrumu je poměrně specifická záležitost, plynoucí zejména z časových omezení scrumu. Testy se samozřejmě provádět musejí a provádějí. Většinou se unit testy nové funkcionality dělají in-line po naprogramování. Bývá ovšem zvykem, že kompletní testy se delegují na testovací tým mimo scrum, nebo se odloží na testovací a debugging sprint (který se vloží mezi nějaké vývojové sprinty).

Tato diplomová práce se dále bude zabývat testováním v širším pohledu, ne na testování ve scrumu, nicméně níže navržená řešení jdou aplikovat i na testování používané ve scrumu.

3.2 Gamifikace testování

Zaměříme-li se nyní na práci testera, jeho práce se skládá ze tří hlavních částí: Vytvořit test, provést test a nahlásit výsledky. Tvorba testu může být delegována na test managera, který testerovi předá vypracovaný test case s jednotlivými kroky (zejména u Usability manuálních testů, viz. 1.1 a 1.1.2.1). V jiných případech je testerovi řečeno, co má otestovat, ale samotný test nebo testovací skript si musí napsat sám.

Při tvorbě testů není příliš co gamifikovat. Jedná se o složitou práci, vyžadující koncentraci sama o sobě. Gamifikováním této části by se do procesu maximálně přidal rušivý element a nevedlo by to k zamýšlenému cíli.

Provádění testů je druhou fází testování. Zde se práce testerů liší podle prováděných testů, viz. 1.1 a 1.1.2 a je velmi těžké generalizovat. Tato práce může zahrnovat důsledné sledování test case, free testing, tvorbu testovacích skriptů a jejich spouštění nebo další činnosti. Na této části také tester tráví nejvíce času, nicméně se jedná o práci poměrně rutinní, jejíž součástí je i čekání na výsledky testů. V této fázi vidím největší potenciál gamifikace.

Reportování výsledků testů je velmi rutinní práce, která ale nezabere příliš mnoho času. Výsledky se typicky sdělují ústně nebo prostřednictvím mailu, jen větší release se testují více formálně včetně nějakých oficiálních výstupů. Teoreticky to vypadá, jako dobré místo pro začlenění gamifikačních prvků, nicméně vzhledem ke krátké době trvání a spoustě různých postupů (jinak se reportují testy uvnitř týmu a jinak zákazníkovi a pod.) zde hrozí riziko přílišné gamifikace, jak bylo naznačeno na konci kapitoly 2.1.

3.2.1 Technická realizace

Gamifikování testovaní se neobejde bez software utilit. Tyto utility obecně mohou být vytvořeny jako standalone software nebo jako modifikace/plugin do samotného testovacího software. Oba přístupy mají své výhody a nevýhody a jsou vhodné pro jiné druhy gamifikace.

Standalone software se vyznačuje tím, že není nijak propojený s testovacím software ani omezený na funkcionalitách. Při vývoji takového software nejsme omezováni dokonce ani účelem, ke kterému bude sloužit. Nevýhodou je, že vyžaduje pro správnou funkčnost více spoluúčasti ze strany uživatele/testera. Samostatný software nemá příležitost, jak komunikovat s testovacím i testovaným software, tedy za import výsledků testů do programu je zodpovědný tester.

Modifikování nebo vytvoření pluginu umožní téměř bezešvou integraci s rodičovským systémem, výsledky testů jsou k dispozici okamžitě, plugin čerpá z již naprogramovaných metod a postupů, zde je ovšem potřeba zkontrolovat dostupné API pro vývoj pluginů, či licenční podmínky upravující práva modifikace software. Obojí může být značně limitující.

Poslední variantou je naprogramovat software, který bere z obou předchozích principů. Jednalo by se o samostatný program ve smyslu, že by měl vlastní spustitelný soubor (oproti pluginu, který se spouští se svým rodičem), ale používal by API testovacího software pro přenos informací. Výhodou tohoto přístupu je, že se již existující a ověřený software dá použít jako backend pro nový program, přičemž ale tento backend nestaví tak zásadní mantinely funkcionalitě, jako v případě pluginu. Nevýhodou je samozřejmě snížení efektivity, neboť nelze dosáhnout takové míry integrace s backendovým systémem. Také není zaručeno, že zvolený backend takové API vůbec odhaluje a pokud ano, je proprietární, tedy pro každý backend by se musel programovat zvlášť adaptér.

3.2.2 Možné způsoby gamifikace

3.2.2.1 Achievements

Achievements (v počeštěné variantě „achievementy“) jsou založené na ocenění hráče (zde testera) za dosažení milníků v průběhu hry, tedy v testovacím procesu. Tyto achievementy mohou být odměnami samy o sobě nebo jejich ocenění může být svázáno s předem definovaným počtem bodů. Využití těchto bodů by pak záleželo na project manažerovi nebo jiném vedoucím pracovníkovi. Příkladem by mohlo být „15 Test Case v řadě bez chyby“.

Nevýhodou tohoto systému je to, že podporuje soutěživost uvnitř týmu, a v závislosti na nastavení achievementů a odměn za ně, by mohl vést i ke snaze zvítězit nad systémem, tedy věnovat se testům, které nejrychleji přinesou potřebný počet bodů, místo toho, co je potřeba otestovat.

Obrázek 3.1: Příklad achievementů ve hře World of Warcraft

3.2.2.2 Inventory

Přístup s názvem „Inventář“ vychází z principu sběratelských her a je obměnou předchozího přístupu. Každý úspěšný test by měl určitou šanci poskytnout uživateli virtuální předmět nebo token. Tyto předměty by šlo využít pro postup ve hře, která by byla nastavena pro vývojový tým.

Oproti systému achievmentů nepodporuje tolik soutěživost, zato více podporuje snahu zvítězit nad systémem. Mohlo by se ale snadno stát, že by tester budoval a pouštěl jen lehce obměněné varianty lehkých testů, aby rychle získal tokeny do inventáře a testování složitějších test case by bylo odsunuto na vedlejší kolej.

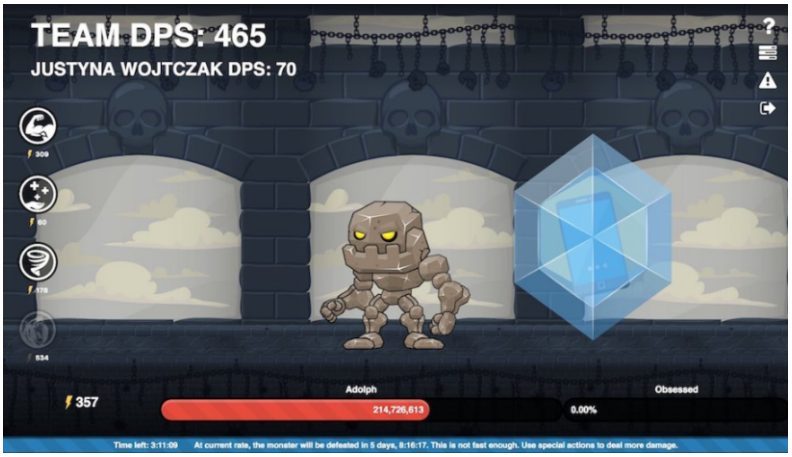
Obrázek 3.2: Příklad využívání nasbíraných věcí v browser hře Mafia Wars

3.2.2.3 Resources

Tato evoluce předchozích principů vychází ze základního kamene strategických her, především populárních browser her. Úspěšné dokončování testových případů by poskytovalo určité množství „surovin“, které by byly poskytnuty centrálnímu virtuálnímu projektu (např. *postavíme raketu*) a jeho dosažení může být projektovým manažerem odměněno dle jeho uvážení.

Tento přístup trpí podobnými neduhy jako přístup Inventář. Narozdíl od inventáře však podporuje spolupráci, protože i suroviny spolupracovníků jsou poskytovány pro dosažení společného cíle. Na druhou stranu, opět je zde nežádoucí efekt v podobě grindu[2]. Mohlo by tak snadno dojít k situaci, kde by testeři zakládali spousty jednoduchých testů místo komplikovaných případů.

[2]Rutinní opakování činnosti, která se ukázala jako nejefektivnější k dosažení cíle.



Obrázek 3.3: Příklad využívání nasbíraných resources (zde DPS) k poražení Bosse a získání odznaku ve hře GetBadges

3.2.2.4 Visualization

Jedná se o odlišný přístup ke gamifikaci testování. Spočívá v zábavné a zároveň efektivní vizualizaci testů a jejich výsledků. Testovací GUI by se přiblížilo point-and-click hrám a sloužilo by jako nový centrální hub pro budování a spouštění testů.

Vzhledem k tomu, že tento přístup pouze nepřebírá výsledky testů, ale aktivně spravuje testy i jejich běh, je zde kladen mnohem větší nárok na integraci gamifikační utility do testovacího software.

Oproti ostatním přístupům ovšem nemění testovací kulturu testerů, nepodporuje žádné pokusy o poražení systému ani soutěživost mezi testery.

3.3 Shrnutí

Existuje množství způsobů, jak přistoupit ke gamifikaci testování software, některé z nich byly nastíněny v předchozích sekcích. Společný problém většiny z nich je fakt, že mění zaběhlé pořádky ve vývojovém týmu. Testování je velmi variabilní proces reagující na okamžité potřeby, nedá se tedy lehce naprogramovat jednoznačný systém checkpointů, který by zajistil, že testeři budou dělat to, co mají, a ne to, co přinese největší virtuální odměny. Navíc mezi mými spolupracovníky převládá dojem, že jakýkoli „sběratelský“ systém není dostatečně zajímavý na to, aby vyvážil problémy spojené s vývojem, nasazením, údržbou a realizací takového systému. Často také zaznívaly stížnosti na to, že „už takhle je to (testovací prostředí JMeter, pozn. autor) dost nepřehledné“. Tyto argumenty mě přiměly zaměřit se na gamifikaci směrem vizualizace, která výše uvedenými problémy netrpí (viz. 3.2.2.4).

Kapitola 4

Proof of Concept

Cílem této části diplomové práce je připravit prototyp aplikace, která by demonstrovala gamifikační prvky pro testovací účely a mohla sloužit jako nový front-end pro testovací software. Hlavní myšlenka vychází z přístupu vizualizace, který byl nastíněný v kapitole (viz. 3.2.2.4).

Tento prototyp nebude implementovat žádné propojení s testovacím softwarem, ani nebude implementovat vlastní testovací prostředí. Demonstrace možností prototypu bude realizována mockováním. Tato diplomová práce se ovšem bude zabývat i možnostmi následné implementace do testovacího prostředí.

Součástí implementace naopak budou všechny funkce, které jsou potřeba nejen ke gamifikaci, ale i k příjemné práci s programem tak, aby mohl nadále plnit své testovací funkce. Při identifikaci takových funkcí a nutností vycházím ze své praxe, přičemž s testovacím softwarem JMeter pracuji téměř denně. Tyto názory jsem dále upevnil konzultací se svými kolegy.

Cílovým uživatelem je programátor, který během své práce využívá program Jmeter nebo podobný. Typicky tedy vývojář serverových aplikací, které s klientem komunikují prostřednictvím JMS, SOAP nebo jiných druhů zpráv.

4.1 Cílový uživatel

Cílový uživatel tohoto dema je takový, který ke své práci využívá v současnosti program JMeter od Apache a nebo konkurenční software. Zaměřuje se tedy především na programátory, kteří vyvíjejí software a potřebují jej otestovat před odevzdáním k zevrubnějším testům (např. integračním nebo systémovým).

Při vyvíjení vycházím z pocitů a potřeb, které jsem získal při své práci. V dalších odstavcích se budu odvolávat právě na JMeter, nicméně obecné závěry vyplývající z této práce, se dají aplikovat i v případě, že uživatel má zkušenosti s jiným testovacím software. Analýza je prováděna z pohledu uživatele v roli podobné té výše zmíněné. Není zaměřena na „koncové“ testery, kteří už

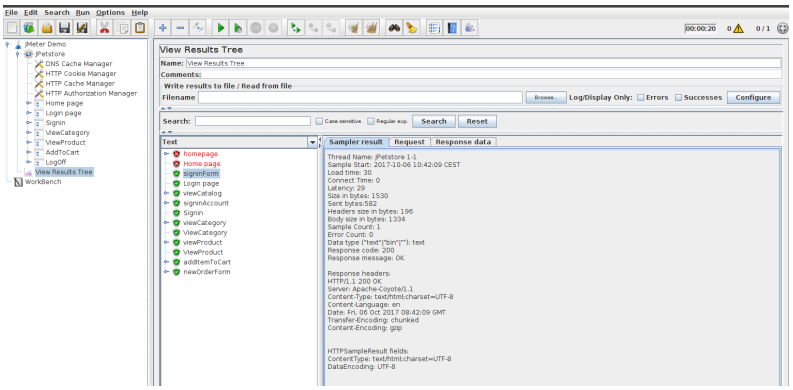
více používají samotnou vyvinutou aplikaci a zkoušejí jednotlivé Test Case z uživatelského pohledu. (viz. 1.1 a 1.1.2.1)

4.1.1 JMeter

Apache JMeter je javový open-source program, navržený pro testování funkcionalit a měření výkonu. Primárně vytvořen pro testování webových aplikací, JMeter nyní zvládá spoustu dalších věcí. JMeter lze v současnosti používat k testování a měření výkonu jak statických, tak dynamických zdrojů, umí simulovat zátěž a komunikuje jak s jednotlivými servery, tak i s clustery. JMeter ovládá množství protokolů a je kompatibilní s množstvím technologií, z nichž nejdůležitější jsou HTTP, SOAP a REST, FTP, JDBC a JMS.[13]

JMeter je vysoce přizpůsobitelný. Téměř každá komponenta je napojena na jádro jako plugin. Komunita může vytvářet další pluginy, které nahradí nebo doplní ty stávající. Které pluginy se nahrají rozhoduje nastavení JMeteru při startu. Díky tomu může každý tester používat JMeter tak, jak potřebuje. Testeři performance testů nejspíše využijí pluginů pro monitoring výkonu a časovače, zatímco testeři funkcionalit pro tyto pluginy nebudou mít využití a spíše sahnou po některém z vizualizačních pluginů.

V této části bych rád demonstroval uživatelské rozhraní JMeteru a některé nedostatky, které spatřuji.



Obrázek 4.1: Okno JMeteru s vybraným výsledkem testů

Na obrázku 4.1[14] je demostrován pohled, který se testerům naskytuje během testování velmi často. Takto vypadá JMeter když zobrazuje výsledek spuštěných testů.

Levá část slouží jako browser vytvořených testů. Existuje několik druhů entit, které dohromady tvoří strukturu browseru. Složky se v JMeteru označují jako Controllery a nabízejí i různé možnosti nastavení kdy nebo jak spustit testy, které jsou v něm obsaženy.

Ve zbytku okna se nachází plugin zobrazující výsledky.

Uvažte následující fakta:

- Názvy složek (controllerů) se nepropisují do výsledků. Výsledky nejsou strukturované, nedají se procházet po složkách. Je to lineární seznam spuštěných testů.
- Vybraný výsledek nelze spojit s testem, který jej vyvolal jinak, než manuálním hledáním v browseru podle jména a podvědomého odhadu, ve které složce se bude nacházet.

Tyto dvě vlastnosti velmi kazí efektivitu JMeteru jako nástroje pro Unit testování. I toto vychází z filozofie tvůrců JMeteru, že jeho UI je zastaralé a JMeter by se měl pro testování používat výhradně jako konzolová aplikace a UI používat jen pro tvorbu testových plánů. Tato filozofie je možná dobrá pro rozsáhlé testování, ale pro lazení testu, zvlášť když se tento test ještě často mění, je naprosto nepraktická.

4.2 Klíčové funkce

Identifikoval jsem následující funkce a požadavky, které musí prototyp splňovat. Prakticky se jedná jen o rozložení modelu, protože většinu funkčních požadavků prototyp nevyužije.

4.2.1 Hierarchie testů

UI bude pracovat s hierarchickým rozdělením testů, přičemž každá úroveň bude mít své vlastní unikátní zobrazení. Z praxe se osvědčilo následující dělení:

1. **Test Suite:** Sada testů, logicky spadající do stejné, obvykle businessové, kategorie.
2. **Test Block:** Blok obsahující testy spadající pod stejnou logickou jednotku.
3. **Test Case:** Konkrétní případ k otestování jedné konkrétní funkcionality.
4. **Test Step:** Jednotlivé kroky, které jsou potřeba učinit pro splnění daného test case.

Konkrétně, pro případ *Chytrého domova*, zástupci jednotlivých úrovní mohou být:

1. **Test Suite:** Testy světel
2. **Test Block:** Testy žárovek, Testy LED pásků
3. **Test Case:** Rozsvícení žárovky, Zhasnutí žárovky, Změna barvy světel

4. **Test Step:** *Konkrétní JMS posílaná serveru* nebo *Database Check*

Tento bod je velmi důležitý nejen z hlediska gamifikačního, ale i praktického. JMeter sice umožňuje strukturovat testy při jejich vytváření, nicméně zobrazení jejich výsledků je už lineární (v jednom seznamu), navíc se z výsledku nejde rychle prokliknout zpět na daný test. To škodí čitelnosti i efektivitě. (viz. kapitola 4.1.1)

4.2.2 Další požadavky pro ostrou verzi

V případě rozvinutí prototypu je zapotřebí uvážit další požadavky a potřeby. Stále se jedná o poměrně krátký výčet, protože koncový program převezme většinu funkcionality z napojeného backendu. Identifikované požadavky jsou pak následující

4.2.2.1 Výkon

Pro prototyp nehraje příliš velkou roli, nicméně pro nasazení s backend testovacím software je zapotřebí počítat s očekáváním a potřebami testerů na rychlost testů. Jádro výkonu stále zůstává samozřejmě na backendu a samotné testované aplikaci, nicméně frontend musí být navržen tak, aby tento výkon nadměrně neomezoval. Nice-to-have funkce související s výkonem je pak zobrazování výsledků v reálném čase. Implementace této funkce je závislá na schopnostech backendového software, případně jeho API.

4.2.2.2 Vlastní paměťový prostor a skriptování

Prototyp momentálně neumožňuje nahrávání proměnných do paměti, či spouštění skriptů. Toto je poměrně omezující, protože není k dispozici žádný generátor hodnot. Pro testování, kde je zapotřebí zadávat unikátní ID, by to znamenalo před každým testem přepsat všechny zprávy. To je časově neefektivní a vysoce obtěžující, navíc zavdává prostor k chybám.

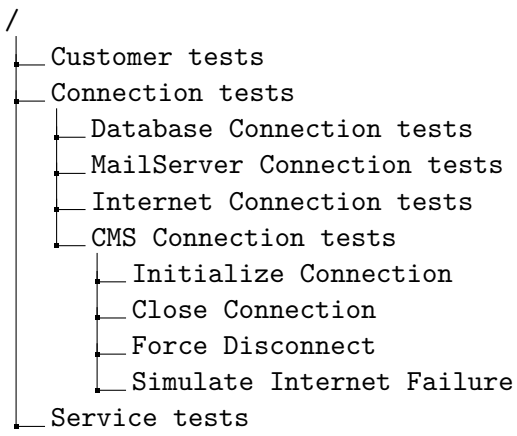
Datová realizace by mohla proběhnout jednoduše statickou mapou o dvojici Název-Hodnota, do které by se načítalo pomocí skriptů. Nicméně musí dojít k úpravě UI tak, aby podporovalo tyto skripty. Pravděpodobně by také muselo dojít k rozšíření počtu potomků Test Step o skript, protože execute scriptu je velmi odlišný od execute zpráv nebo dotazů do databáze.

4.2.2.3 Konfigurovatelnost

Spolu s připojením k testovacímu software je třeba využít interface tak, aby bylo možno nastavovat konfiguraci z UI. Tato konfigurovatelnost se týká zejména počtu vláken a hlavně IP adres cílových programů a databází. Tyto hodnoty nejdou z prototypu nastavit, jejich konfigurovatelnost bez nutnosti přepnout se do backend software je ale žádoucí.

4.3 Use Cases

V této části popíši dva vzorové use case pro tento prototyp. Pro tuto sekci předpokládám následující hierarchii testů:



4.3.1 Use Case: Spustit testy

Uživatel chce spustit všechny testy v souboru. Na to je zapotřebí těchto kroků:

1. Spustit program
2. Načíst soubor s testy
3. Kliknout na tlačítko „Run“ v záhlaví

Po spuštění programu a načtení žádaného souboru je vidět seznam všech Test Suite ve svých neutrálních stavech a záhlaví. V záhlaví uživatel klikne na tlačítko Run.

Testy se začnou spouštět. V závislosti na počtu a druhu testů, kvalitě testovaného software a dalších okolnostech to může trvat i velmi dlouhou dobu.

Po dokončení testů budou všechny ikony reprezentující Test Suite nastaveny buďto do úspěšné nebo neúspěšné.

4.3.2 Use Case: Spustit test na Force Disconnect od CMS Serveru

Uživatel chce ověřit funkcionalitu vynuceného odpojení od CMS Serveru. Na to je zapotřebí těchto kroků:

1. Spustit program
2. Načíst soubor s testy
3. Kliknout na tlačítko Test Suite „Connection tests“

4. Kliknout na tlačítko Test Blocku „CMS Connection tests“

5. Kliknout na tlačítko Test Case „Force Disconnect“

6. Kliknout na tlačítko „Run“ v záhlaví

Po spuštění programu a načtení žádaného souboru musí tester projít skrz hierarchii na Test Case „Force Disconnect“. V záhlaví stiskne tlačítko Run.

Testy se začnou spouštět. Test Case je jednoduchá jednotka testů, měla by být do minuty hotová. Opět závisí na různých okolnostech.

Po dokončení všech Test Stepů bude ikona Test Case i ikony všech podřízených Test Stepů nastavena na úspěch nebo neúspěch. Ostatní testovací jednotky zůstávají v neutrálním stavu.

4.4 Technické provedení

4.4.1 Programovací jazyk

Pro vývoj prototypu se nabízí využití několika programovacích jazyků a přidružených technologií. Většina testovacích programů funguje jako desktopová aplikace, frontend tedy musí být také koncipován jako desktopová aplikace. Ze všech možností jsem vybral dvě, které podrobím detailnějšímu pohledu. Jedná se o Javu a C#.

4.4.1.1 Java

Java je v současnosti jedním z nejpoužívanějších jazyků. Java je hojně využívána nejen jako technologie pro vývoj podnikových aplikací, ale i pro podpůrné utility, jako je právě třeba testovací JMeter. Výhodou Javy je také to, že ve svých development kitech už obsahuje nástroje pro serverové aplikace, messaging, konektivitu, databázový přístup (i když zde je občas odkázána na externí drivery v závislosti na DB) a rovněž i její multiplatformnost. Vyvinutá aplikace bude fungovat beze změn v kódu pro počítače jak s OS rodiny UNIX, tak i na počítačích s OS Windows, tedy za předpokladu, že obě prostředí mají nainstalované shodné runtime prostředí.

4.4.1.2 C# a .NET

C# je objektově orientovaný jazyk z rodiny jazyků C. V dnešní době je C# velmi často zmiňován v tandemu s .NET frameworkem. Tato platforma je vyvíjena a udržována společností Microsoft, která také zaručuje podporu. .NET je součástí OS Windows a to od verze Windows 7 a novější. Starší Windows samozřejmě podporují .NET také, ale bylo potřeba je manuálně doinstalovávat. Díky této integraci je chod .NET aplikací prakticky zaručen na všech zařízeních s Windows 10, které procházejí updaty.

Integrace .NET aplikací mimo PC s Windows (či Windows Server servery) byla vždy problematická. Microsoft dlouho nevyvíjel framework pro konkurenční OS. To se změnilo v roce 2016 s vydáním .NET Core[15]. Core verze .NETu je opensourcová a funguje i pro UNIXová prostředí. V současné době je framework .NET Core ve verzi 2.0 a blíží se funkčnosti původního .NET.

4.4.2 Datová perzistence

Struktura vytvořených testů musí být někde uchována. Pro tento účel se nabízí hned několik přístupů:

4.4.2.1 Využití backendových struktur

Prototyp je sice zamýšlen jako samostatná aplikace s mockovaným backendem, ale živá aplikace by už využívala jeden z funkčních testovacích programů. Tyto programy samozřejmě již mají vymyšlené, jak uchovat testy mezi jednotlivými spuštěními instance softwaru. Například JMeter využívá vysoce strukturované XML.

Nevýhodou tohoto přístupu je to, že každý testovací program má svůj proprietární formát, který není kompatibilní jeden s druhým. Pro každý podporovaný software by se tedy musel napsat vlastní parser, což by mohlo být značně pracné, obzvláště když uvážíme, že každý projekt používá jeden testovací software a je velmi nepravděpodobné, že by začal pracovat s jiným, parsery navíc by tedy byly jen „mrtvým“ vývojovým časem.

4.4.2.2 Vlastní plaintextový soubor

Tento přístup vychází z předchozího. Místo využívání již existujících formátů by se ovšem vytvořil nový formát pro uchování testů spolu s jedním parserem. Tento formát by vycházel z XML, neboť většina programovacích jazyků má způsoby, jak pracovat s XML soubory.

Tento přístup dědí výhody i nevýhody XML souborů u tradičních testovacích programů. Mezi výhody patří zejména dobrá přenositelnost. Nevýhodou je nutnost tento soubor verzovat, pokud k testům přistupuje více než jeden uživatel. Z vlastních zkušeností vím, že dochází velmi často ke konfliktům ve verzování, což vede k tomu, že každý tester začne používat svůj vlastní soubor a testy se musejí jednou za čas synchronizovat.

4.4.2.3 Databáze

Jedná se o neotřelý přístup k ukládání testovacích dat. Pravděpodobně by se dalo argumentovat, že se jedná o zbytečně složité řešení, které sebou ale nese nezanedbatelné množství výhod. Data mohou být ukládána strukturovaně. Filtrování je rovněž rychlejší nad databází, než nad plaintextem. Data v databázi jsou nepochybně lépe zabezpečená, než plaintextová. Rovněž data

jsou prakticky okamžitě synchronizována, většina konfliktů při současné práci více lidí najednou nehrozí.

Nevýhodou je pak samozřejmě jistá míra neflexibility. Data nejsou tak snadno přenositelná a je třeba pracovat z místa, odkud je přístup k databázi. Vzhledem k tomu, že testeři obvykle pracují z kanceláře a testy se málokdy spouští „na cestě“, není tato nevýhoda příliš omezující a pro účely home office se dá využít vzdálený přístup nebo VPN.

4.4.3 Cílový systém

V dnešní době existují prakticky tři rodiny operačních systémů, pro které se dá vyvíjet software. Samozřejmě se jedná o Microsoft (Windows a Server), UNIX a OSX. Poslední zmíněný sice vychází technologicky z UNIXu, ale obsahuje velké množství specifických vlastností, že je nutné jej uvést jako separátní entitu.

Microsoft Windows je standardním OS většiny desktopů a i velké množství programátorů vyvíjí právě pod MS Windows. Většina testovacích software má také své Windows varianty. Přestože kompletní vývojový cyklus může být dokončen pouze za použitím UNIXových prostředků, mám zkušenosti ze studií i zaměstnání, že většina GUI práce se odehrává právě na OS Windows, zatímco UNIX je spíše doménou serverů.

OSX je proti ostatním systémům menšinově zastoupený mezi programátory. Kromě toho, že počítače Mac důstojně zastoupí UNIXové (a s jistými omezeními i Windowsové) počítače, jsou to také vzhledem k politikám společnosti Apple jediné systémy, na kterých se dá programovat pro ekosystém Apple (OSX a iOS).

4.4.4 Rozhodnutí

Rozhodl jsem se vytvořit prototyp v Javě. Pro Javu jsem se rozhodl ze dvou hlavních důvodů. Prvním je to, že Java je jako jazyk velmi flexibilní. Takto vytvořený prototyp bude k dispozici k sestavení jak pod Windows, tak pod UNIXem, tak i pod OSX. Druhý důvod je to, že nejpoužívanější testovací software jsou rovněž napsány v Javě. Použití stejného jazyku usnadní případnou integraci s backendovou funkcionalitou. Fuknčnost bude testována a demonstrována na stroji s MS Windows.

Pro uchovávání dat jsem dlouho uvažoval nad přístupem pomocí databáze. Nakonec jsem se rozhodl pro tradiční postup formou plaintextového .xml souboru. Toto rozhodnutí opírám zejména o fakt, že programátoři/uživatelé jsou zvyklí ukládat své testy do souborů, přičemž oddělují jednotlivé „důvody“ pro testování. Kompletní testy tedy mohou mít jiný soubor než testy na rychlé ověření nových funkcionalit a není nutné načítat všechny testy do paměti.

Kapitola 5

Realizace prototypu

5.1 View - UI

Pro tvorbu UI je použit javový framework swing. To umožňuje využívat různých formulářových výhod, jako jsou layouty a jednoduché odchytávání eventů.

Hlavní okno je tvořeno základním formulářem, na kterém jsou vykreslovány jednotlivé úrovně testů. Testy mají také své vlastní okno s detaily. Tématem klipartů je „kancelář“, více informací u jednotlivých úrovní.

5.1.1 GUI

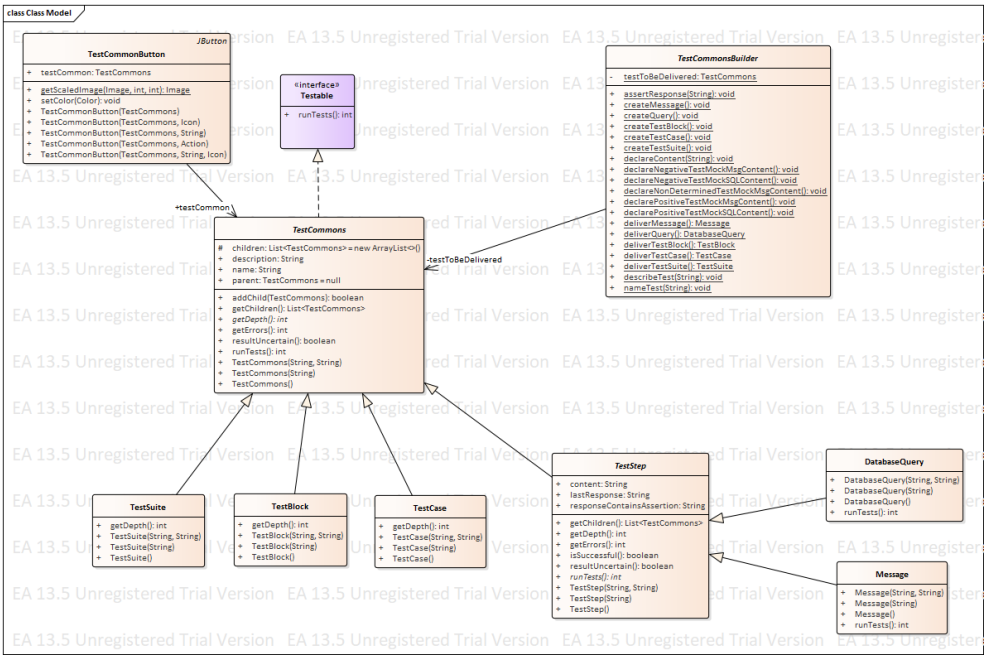
View je tvořený jedním fyzickým oknem, které je rozděleno na dvě části - záhlaví a hlavní plochu. Na hlavní ploše se zobrazují jednotlivé testovací entity, ty budou vysvětleny v sekci 5.2. Záhlaví je tvořeno text boxem, ve kterém je napsáno jméno konkrétního vybraného testu, a tlačítky pro ovládání programu.

1. Back : Vrátí vybraný test o úroveň výš
2. Run : Spustí právě vybraný test
3. Details : Zobrazí další okno s popisem testu
4. Add : Přidá další pod-test vybranému testu

Tato horní část je nastavená tak, aby se škálovala do šířky roztažením text boxu se jménem. Na výšku dochází jen k minimálnímu škálování. Hlavní část je nastavená, aby se škálovala jak horizontálně, tak vertikálně. Ovladací prvky se neškálují, zvětšení okna zvětšuje prostor na zobrazení více testů.

5.2 Model

Model se skládá z tříd, které reflektují výše stanovenou analýzu. Jádrem celého modelu je kaskáda testové hierarchie. Tato byla implementována pomocí modifikovaného Composite patternu (viz obr. 5.1), kde třída Composite byla nahrazena třemi třídami stanovené dle hierarchie. To umožňuje lehce zajistit, že jako podřízené části budou vždy správného podtypu. Funkci listu zde plní Test Step, který je abstraktní rodičovskou třídou pro Message a DatabaseQuery. Toto rozdělení není pro demonstrační účely potřeba, ale v reálných situacích se databázové dotazy vykonávají jinak než posílají zprávy, proto jsou zde rozděleny.



Obrázek 5.1: Datový model prototypu. Neobsahuje třídy realizující okna a mock

5.2.1 Test Suite

Entita označovaná jako Test Suite označuje nevyšší jednotku testové hierarchie. Je určena pro velké logické jednotky testů, jako například *Customer tests*, *Connection tests*, *Service tests*. Jedná se o businessové třídění a nemělo by sloužit k dělení na různé druhy testů (viz. 1.1).

Test Suite by neměly být *FucntionalTests Test Suite*, *PerformanceTests Test Suite* nebo *IntegrationTests Test Suite*. Každý z těchto druhů by měl mít svůj vlastní soubor (eventuálně i s vlastním nastavením).

Spuštění testů nad Test Suite znamená spustit testy pro všechny Test Blocky, které tento Test Suite obsahuje. Vyhodnocení výsledku probíhá agregací počtu chyb z těchto Test Blocků. Pokud některý Test Block obsahuje chybu, je celý Test Suite prohlášen za obsahující chyby.

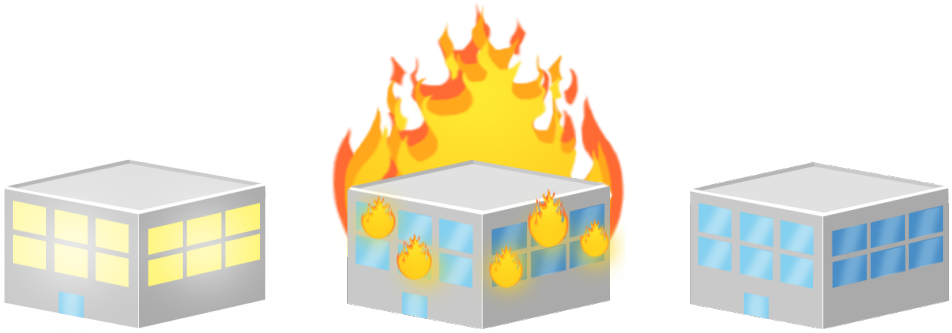
5.2.1.1 Logický význam

Test Suite má především význam pro zásadní oddělení testů, které mohou obsahovat podobné procesy. Pokud tedy dojde ke změně základních procesů (např. zákazník dostane nový data model), tester má k dispozici jednotné místo, kde může tuto změnu a funkcionalitu ověřit. Test Suite se zákaznickými testy bude mnohem zevrubnější v testování všelijakých operací nad zákazníky než jiné Test Suite.

5.2.1.2 Grafická reprezentace

Test Suite je v programu reprezentován jako kancelářská budova. V základním stavu stojí budova s vypnutými světly a čeká, až nad ní budou spuštěny testy.

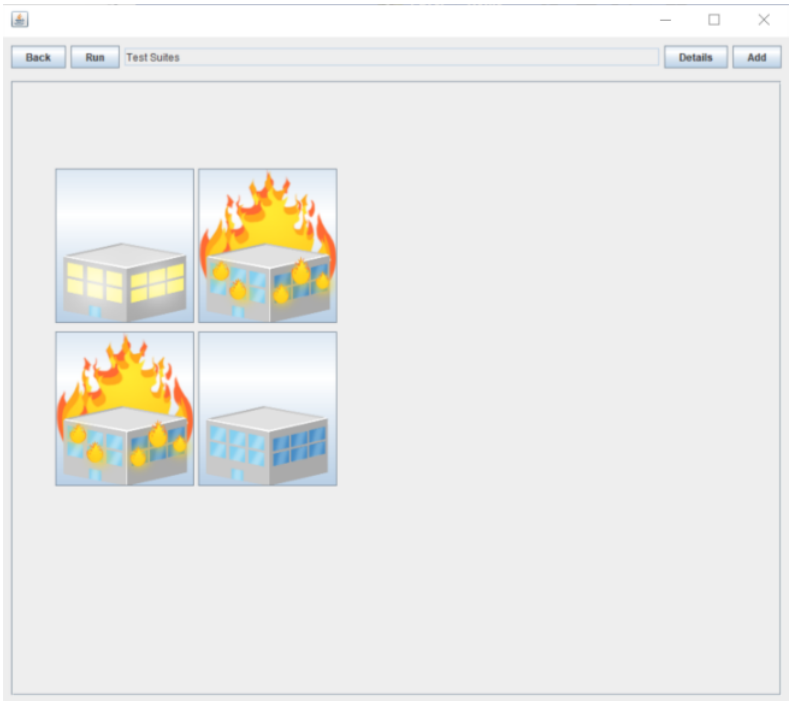
Po spuštění a vyhodnocení testů je fukncní Test Suite (všechny podřízené testy proběhly dobře) reprezentován jako budova s rozsvícenými okny. V Test Suite budovách, kde došlo k chybám, naopak vypukne požár, viz. obr 5.2.



Obrázek 5.2: Cliparty reprezentující Test Suite. Zleva: Test Suite bez chyb, Test Suite obsahující chyby, Test Suite bez známého výsledku

Obrázek 5.3 pak zahycuje použití těchto ikon v programu. Na tomto obrázku je zachycen testovací soubor obsahující čtyři Test Suite, z nichž jeden vykazuje úspěch, dva neúspěch a jeden je v neutrálním stavu, protože nebyl spuštěn.

Rovněž vrchní řádek popisující stav obsahuje pouze informaci *Test Suites*. Je to z toho důvodu, že žádná testovací entita ještě nebyla vybrána a proto je zmíněn pouze obecný popis.



Obrázek 5.3: Okno programu znázorňující výběr Test Suite. Spodní pravý Test Suite nebyl spuštěn, proto je budova zhasnutá.

5.2.2 Test Block

Test Blocky jsou testovací entity podřazené Test Suite. Test Blocky jsou koncipovány jako container jednotlivých Test Case pro konkrétní business potřebu. Například pro Test Suite *Connection Tests* můžou být odpovídající Test Blocky *Database Connection Tests*, *CMS Connection Tests*, *MailServer Connection Tests*

Spuštění testů nad Test Blockem znamená spustit všechny podřízené Test Case. Test Block nahlásí, že proběhl v pořádku jen pokud všechny jeho Test Case proběhly v pořádku.

5.2.2.1 Logický význam

Test Blocky slouží k zahrnutí všech Test Case týkající se jednoho konkrétního business objektu do jednoho místa. Myšlenka za nimi je taková, že pokud tester prověří Test Block týkající se daného objektu, může prohlásit, že objekt funguje správně.

5.2.2.2 Grafická reprezentace

Po zvolení Test Suite se v programu zobrazí seznam Test Blocků, které tento Test Suite obsahuje. Každý takový Test Block je koncipován jako tlačítko s popisem. Tlačítka jsou rozmístěná ve dvou sloupcích a evokují panel výtahu.

Po spuštění a vyhodnocení testů je fukční Test Block reprezentován zeleným tlačítkem. Chybující Test Blocky pak červeným, viz. obr 5.4.

Obrázek 5.4: Cliparty reprezentující Test Block. Zleva: Test Block bez chyb, Test Block obsahující chyby, Test Block bez známého výsledku

Obrázek 5.5: Okno programu znázorňující výběr Test Blocku. Poslední Test Block nebyl spuštěn.

5.2.3 Test Case

Test Case je základní, i když ne nejmenší, jednotka testování. Test Case reprezentuje jeden konkrétní úkol, který se na objektu testuje. Například pro Test Suite *Connection Tests* a Test Block *CMS Connection Tests* mohou být příslušející Test Case *Initialize Connection*, *Close Connection*, *Force Disconnect* a *Simulate Internet Failure*.

Spuštění Test Case znamená spustit všechny podřízené kroky - Test Step. Test Case nahlásí, že proběhl v pořádku, pokud v žádném kroku neselže.

5.2.3.1 Logický význam

Test Case slouží k prověření konkrétní funkcionality pro konkrétní objekt. V testovacích reportech jsou Test Case obvykle nejnižší jednotkou, která test managery zajímá. Rovněž v tomto programu je Test Case nejmenší *logická* jednotka - Tedy taková, kterou může test manager rozhodnout, že jí chce nebo nechce testovat. Pro účely spouštění testů se Test Case skládá z jednotlivých kroků, které je potřeba projít, aby byla funkcionalita Test Case ověřena, nicméně jejich struktura je diktována testovaným programem.

5.2.3.2 Grafická reprezentace

Po zvolení Test Blocku se v programu zobrazí seznam Test Case, které tento Test Block obsahuje. Každý takový Test Case je reprezentován jednou stick figure, která má evokovat zaměstnance na daném „Test Blockovém podlaží“.

Test Case bez posledního známého výsledku je reprezentován zmatenou postavou s pokrčenými rameny. Po spustění testů nad Test Case je poslední známý výsledek stanoven a Test Case je reprezentován buď úspěšnou postavou dávající thumbs up - zdvižený palec, nebo postavou držící se za hlavu, v případě, že Test Case selhal, viz. obr 5.6.

Obrázek 5.6: Cliparty reprezentující Test Case. Zleva: Test Case bez chyb, Test Case obsahující chyby, Test Case bez známého výsledku

Obrázek 5.7: Okno programu znázorňující výběr Test Case. „Zaměstnanci“ jsou seřazeni v řadě, k nedostupným se dá dostat scrollováním.

5.2.4 Test Step

Test Step je nejmenší testovací jednotka programu a zároveň list celého composite patternu v modelu. Test Step představuje konkrétní krok, který je třeba učinit, aby Test Case, ke kterému je tento krok přiřazen, mohl postoupit. Reálně se jedná o řadu (JMS) zpráv pro testovanou aplikaci doplněnou o databázové query. Každý Test Step umí přijmout odpověď od serveru a lze nastavit assertion na string, který bude tato response obsahovat. Pro demonstraci jsem rozdělil Test Step na třídy Message a Database Query. Pro plnou funkcionalitu by bylo vhodné zavést i další typy Test Stepů nebo další funkcionalitu k těm současným (např. možnost uložit výsledek response do paměti, místo porovnávání s assertion). Toto je ale nad rámec vyvíjeného prototypu.

Spuštění Test Stepu znamená odeslat nastavený Test Step do backendového software, který se postará o zbytek zpracování dotazu až po obdržení a předání odpovědi zpět do programu.

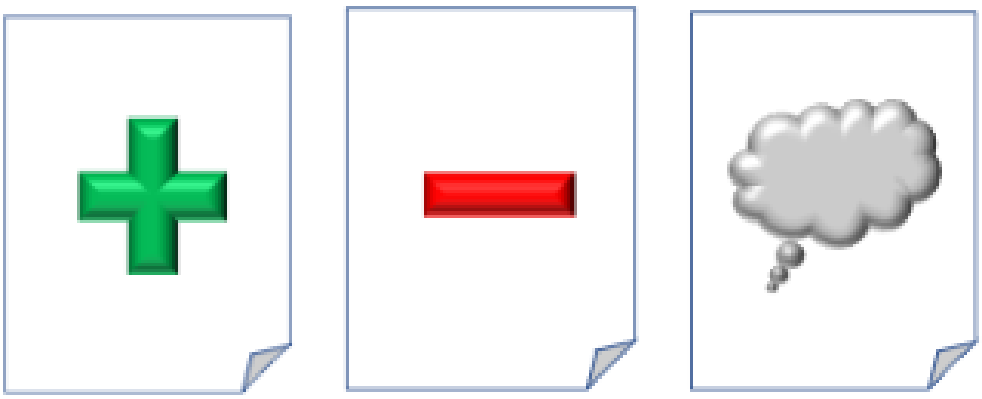
5.2.4.1 Logický význam

Test Stepy slouží jako jednotlivé příkazy - stavební kameny, ze kterých se sestává Test Case. Jejich zavedení bylo nezbytné, neboť většinu Test Case nelze provést na jednu zprávu.

5.2.4.2 Grafická reprezentace

Po zvolení Test Case se v programu zobrazí seznam kroků k vykonání tohoto Test Case. Každý Test Step je reprezentován ikonou formuláře, to má reprezentovat jednotlivé úkoly, které tento zaměstnanec má. Na rozdíl od ostatních testových entit, zvolení Test case neodstraní z obrazovky stick ikony reprezentující Test Case, namísto toho se jednotlivé Test Stepy objeví v další řádce.

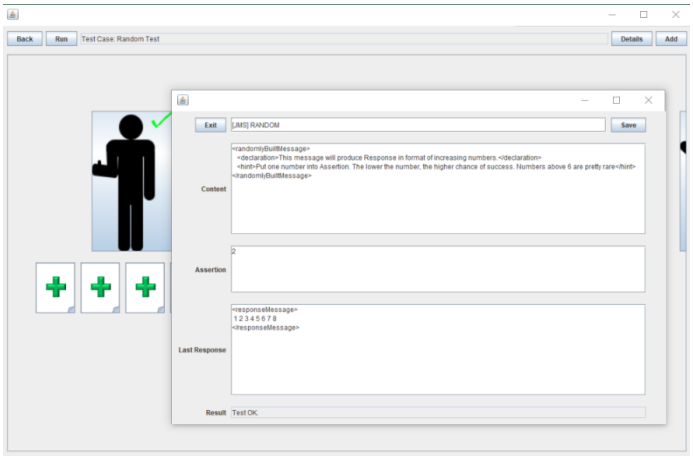
Test Step bez posledního známého výsledku je reprezentován formulářem s oblakem. Výsledek testu se vyhodnotí okamžitě po obdržení odpovědi. Pokud odpověď obsahuje substring, který má tento krok uložený v předpokladu (assertion), krok je vyhodnocen jako správný a formuláři je přidělena ikona zeleného plus. Ke stejnému závěru dojde, pokud není předpoklad vyplněn. Pokud odpověď neobsahuje v sobě string z předpokladu, je vyhodnocen jako chybný a formuláři je přidělena ikona červeného mínus. Viz. obr 5.8.



Obrázek 5.8: Cliparty reprezentující Test Step. Zleva: Test Step se správným předpokladem, Test Step s chybným předpokladem, Test Step, který ještě v runtime neobdržel odpověď.

Obrázek 5.9: Okno programu znázorňující výběr kroku. „Zaměstnanci“ v řadě zůstali, v řadě pod nimi se objevil seznam úkolů, který k tomuto „zaměstnanci“ patří.

Rozkliknutí Test Stepu vyvolá formulář s nastavením tohoto stepu. K dispozici je nastavení zprávy, kterou odešle a assertion. Dale je k dispozici poslední obdržená odpověď na tuto zprávu, nicméně tato není editovatelná, je vyplňována za běhu programu, tak jak přicházejí odpovědi.



Obrázek 5.10: V popředí detail testu. V pozadí hlavní okno programu s Test Stepy.

5.3 Technické schopnosti prototypu

Prototyp je realizací naprostého minima funkcionality, která je zapotřebí pro demonstraci nápadu gamifikace testování software. Účelem této práce nebylo vyvinout funkční řešení připravené k nasazení, ale ukázat cestu, kterou by se mohli další programátoři ubírat. Prototyp se zaměřil zejména na dvě důležité věci:

První je průchod skrz testovací jednotky. Každá úroveň má svou vlastní stránku. Pro urychlení práce by se mohl přidat browser list, ale není nezbytný, neboť do rootu se člověk dostane na ne více než 3 kliknutí. V porovnání s JMeterem je v implementaci prototypu možné okamžitě modifikovat test v reakci na obdrženou odpověď, zatímco v JMeteru je nutné v seznamu odpovědí identifikovat controller, který ke zprávě patří, najít ho v browseru a tam jej upravit. I bez jakýchkoli gamifikačních prvků je toto výrazné zlepšení v ergonomii a zrychluje případné potřebné opravy testů.

Druhá věc je pak samotné naznačení, jak by mohla probíhat vizualizace těchto testů. Grafika v prototypu je jen základní, neboť účelem bylo naznačit možnosti a demonstrovat principy zmíněné v této diplomové práci. Hierarchie, její zobrazení a interpretace byla již vysvětlena v této kapitole. Oproti JMeteru prototyp poskytuje jednu výraznou vizualizační výhodu, a to tu, že je na první pohled zřejmé, zda testovací jednotka vykazuje chyby nebo ne, a to i na vyšších úrovních než na té nejnižší - na úrovni zprávy.

Kapitola 6

Doporučení do budoucna

V této kapitole se pokusím shrnout doposud získané poznatky a navrhnout, jak tyto poznatky lze využít v případě rozpracování do větších detailů.

V této analýze se opět zaměřím výhradně na JMeter od společnosti Apache.

6.1 Propojení s existujícím software

V předchozích kapitolách byly často zmíněny pojmy jako *napojený backend*, *připojený testovací software* a podobné. Přestože žádný podobný software nebyl do prototypu napojen, pokusím se v této kapitole nastínit několik možností, jak toto propojení realizovat. Některé z možností byly naznačeny v kapitole 3.2.1, tato kapitola tyto myšlenky rozvede a přidá další.

6.1.1 Standalone Software

Vybudování standalone software je naivní přístup a rovněž nejsložitější a nejdražší. Vyvinout vlastní program samozřejmě nabídne nejvíce možností realizace, neomezuje ani v technologii, ani v přístupu, ani v možnostech. V extrémním případě programátor může vytvořit doslova 3D adventuru za použití typicky herních enginů, jako je například Unity. Protože se program píše od začátku, představivosti se meze nekladou a podoba takového programu může být libovolná. Problémem tohoto přístupu je ovšem to, že je nutné naprogramovat veškeré komunikační mechanismy, které propojí testovací a testovaný software. Je třeba myslet na podporu více vláken, síťování, velké množství komunikačních protokolů a dalších vlastností, to vše při zachování relativně nízkých požadavků na hardware počítače, protože pracovní stanice typicky nemívají velký výkon, obzvláště v oblasti grafických možností.

Naprogramování celého backendu není triviální záležitost. Zdrojové kódy JMeteru podle nástroje Locmetrics[16] obsahují 224210 řádků kódu k 11.4.2018 a ověření komunikačních možností s různými druhy testovaných programů si vyžádá spoustu času, který se promítne do ceny celého vývoje.

6.1.2 Export-import-export-import

JMeter jako standalone řešení podporuje perzistenci testovacích souborů. Tyto soubory vytváří sám JMeter a disponují proprietární koncovkou *.jmx*. Tento soubor je ovšem velmi lehce otevřitelný v textových editorech jako je notepad, což odhalí, že jmx soubor je obyčené XML se specifickou strukturou. Více informací o formátu je k dispozici na wiki JMeteru[17].

Důležité je, že jmx soubor je plaintextový soubor. Jako takový není zašifrovaný ani binární, je tedy velmi jednoduché jej modifikovat i mimo JMeter. JMeter rovněž obsahuje možnost spustit jej v non-GUI módu, tedy v režimu bez grafického rozhraní s načteným souborem. Kombinací těchto dvou faktů využívá tento přístup.

Stejně jako v předchozím modelu, grafické rozhraní je prakticky divokou kartou pro programátora. Opět může vzniknout jakkoli propracovaný nebo naopak jednoduchý frontend. Rozdíl je však v tom, že spuštění testů způsobí vytvoření a uložení jmx souboru. Z frontendu rovněž dojde ke spuštění JMeteru s vytvořeným souborem předaným v parametru. Takto předaný soubor bude zpracován JMeterem dle standardního chování. JMeter výsledky vypisuje do HTML souboru. Tento soubor poskytuje velmi dobrou vizualizaci už sám o sobě, nicméně frontend může tento soubor zachytit a provést parse HTML a získané informace použít ve frontendu. Oproti řešení v 6.1.1 zde ovšem je zřejmé omezení rozvoje funkcionality, neboť aplikace využívá neupraveného JMeteru k provádění testů. Front end tedy nedokáže nic, co by neuměl samotný JMeter.

Další nevýhodou je pak absence aktivního spojení mezi těmito programy. Komunikace probíhá prostřednictvím zapisovaných souborů, nedá se tedy v žádném případě označit za realtimové. Směrem do JMeteru to tolik nevadí, nicméně program pak musí čekat, až JMeter dokončí všechny své testy a vypíše soubor, než bude schopný parsovat data do UI. Na základě svých zkušeností vím, že středně rozsáhlé testy mohou trvat desítky minut na dokončení a během tohoto času by program neměl informace o žádném, i již dokončeném, testu.

6.1.3 Plugin

Další krok v integraci řešení blíže k testovacímu software. JMeter podporuje pluginy sepsané v Javě a poskytuje množství abstraktních s nejrůznějšími controly vycházející z typicky Javovských swing controlů. Nicméně komunita kolem JMetru nedoporučuje vývoj vizualizérů[3], protože panuje koncensus, že GUI JMeteru by mělo sloužit zejména pro připravu testů, ne jejich spouštění, obzvláště ve větším množství (a připomenou to při každém zapnutí JMeteru). API je ovšem stále k dispozici, je tedy možné ignorovat JMeter komunitu a

[3]Komunita označuje za vizualizér jakoukoli komponentu, jejíž cílem je zobrazit získaná data v GUI

vyvinout vlastní vizualizér. Tento vizualizér bude ovšem velmi těsně spjatý s JMeterem a funkčně vychází ze swingových JForms, podobně jako ukázkový prototyp. Zde je už programátor poměrně omezen ve svých možnostech a nemůže úplně propustit uzdu své fantazii. Existuje i možnost nevázat se na vnitřní interface vizualizéru a vyvinout více nezávislý plugin, který nebude založený na JForms, ale i tak je programátor omezen výběrem technologie (musí být Java shodné verze s JMeterem, v současnosti Java 8) a třídami, které poskytuje JMeter rozhraní.

Integrace pomocí pluginu rovněž znamená se „upsat“ JMeteru (nebo jinému konkrétnímu software). Zatímco u standalone programu se dilema kompatibility neřeší vůbec a u předchozí metody stačí vyměnit konfiguraci např. abstraktní továrny nebo jiného patternu, který zajišťuje budování souboru, plugin už aktivně využívá prvky rodičovského programu a přestože základní funkcionalita je stejná, implementace se mohou lišit tolik, že migrace pluginu mezi jednotlivými testovacími programy bude obtížná.

6.1.4 Změny v programu

Jedná se o ještě hlubší integraci řešení do testovacího software. JMeter je open source program, jehož zdrojové kódy jsou veřejné. Licence JMeteru je Apache License Version 2.0, jedna z nejpermisivnějších licencí, co se týče úprav zdrojového kódu. Programátor může s JMeterem udělat prakticky cokoli. Tento přístup není vhodný, měl by sloužit spíše jako příklad, čemu se vyhnout, a to především z následujících důvodů:

JMeter je dlouhodobě vyvíjený program lazený po mnoho iterací. První verze JMeteru byla vydána už roku 1999[13], má tedy historii téměř dvaceti let vývoje, ve kterém se s gamifikovaným vizualizérem příliš nepočítalo. Úpravy by si vyžádaly extenzivní analýzu současných zdrojových kódů a poté pozměnit samotný engine tak, aby podporoval to, co programátor zamýšlí. Na každém kroku přitom bude narážet na to, že všechny předchozí iterace sledovaly úplně jinou designovou filozofii. V konečném důsledku se pracnost nejspíš vyrovná pracnosti naprogramování úplně nového standalone software, nebo ji dokonce překoná.

Druhým problémem je samotná licence JMeteru. I pokud by se někdo rozhodl vložit čas a potřebné zdroje do vývoje gamifikační utility změnou JMeteru, bude vázán licenčním ustanovením Apache 2.0 a bude muset dát svůj kód k dispozici. To ztěžuje případné komerční využití. Obecně se toto řešení jeví jako příliš složité a finančně neobhajitelné.

6.2 Ekonomické zhodnocení

Použitelnost této studie po ekonomické stránce záleží na způsobu, kterým se vývojář rozhodne k vizualizaci přistoupit. Při rozhodování je zapotřebí zvážit, jaké ekonomické výsledky čekáme a počítat s velmi volnou licencí Apache 2.0

pod kterou je šířený původní JMeter. Tým, který vyvine podobnou funkci pro sebe nebo příbuzné týmy v společnosti nejspíše nebude tolik řešit fakt, že musí zveřejnit kódy komukoli, jako firma, která by podobné řešení vyvinula pro komerční účely.

6.2.1 Modifikace JMeteru

Jak už jsem zmínil v sekci určené popisu tohoto přístupu, nevidím velký ekonomický potenciál. Modifikovaný JMeter (dále pracovní název ModMeter) by se prakticky nemohl prodávat, protože každý kupující by mohl zveřejnit a šířit dílo dál, ať už zdarma, nebo komerčně za nižší částku, neboť by nemusel pokrývat značné náklady spojené s vývojem.

Jedním z možných modelů monetizace tohoto přístupu by bylo zpřístupnění assetů na základě nějakého předplatného (subscription model) nebo trvale prostřednictvím „store“ (virtuální obchod, kde je možné nakoupit si sadu ikon nebo téma do programu). V tomto případě by nevadilo, že by se ModMeter nemohl prodávat, neboť bez přístupu k assetům by nefungoval, potažmo nepřinášel výhody oproti standardnímu JMeteru. Reverse engineering formátu ikon a nastavení by pak byl pro případné zákazníky, kteří by chtěli obejít náš obchod, natolik ekonomicky náročný, že věřím, že by raději koupili nové sady ikon nebo téma oficiální cestou.

Přesto si myslím, že vytvoření ModMeteru není ekonomicky obhajitelné. Reverse engineering JMeteru, rozsáhlé modifikace jádra a implementace nových featur si vyžádají velké množství času, které se musí zaplatit a obávám se, že obchod s assety by nevygeneroval dostatečný zisk v rozumné době.

Pro nekomerční využití pak tento přístup nedává žádný smysl.

6.2.2 Plugin

Vývoj pluginu je značně jednodušší než zásahy do kódu. Vyvinout plugin zabere citelně méně času a tím pádem i zdrojů. Zde by se eventuálně mohlo vyplatit jít komerční cestou prostřednictvím store, která byla naznačena v předchozí části.

Nicméně vývoj pluginu naráží na limity, které byly popsány v části 6.1.3. Konkrétně se vývojář musí spokojit s grafickými možnostmi podporovanými třídou JForm.

6.2.3 Export-import-export-import

Tento standalone přístup k integraci JMeteru kombinuje volnost standalone programu s využíváním již naprogramovaných funkčních postupů JMeteru. Přestože tento přístup nabízí nejspíš největší volnost za nejméně peněz (odpadá programování backendu kromě dvou parserů), v čisté implementaci nezvládne responzivní chování a výsledky ukáže až po dokončení všech testů.

Toto je vzhledem k ostatním programům včetně čistého JMeteru výrazný funkční krok zpět a nedovedu si představit, že by se takto fungující program mohl uchytit jako komerční řešení.

Upravená verze tohoto přístupu pak počítá kromě vývoje samostatného programu i s úpravami do JMeteru, které by eliminovali tuto chybu. Jistě to prodlouží vývojovou dobu i cenu vývoje, nicméně řešení bude konkurenceschopné. Rovněž je to první a nejlevnější řešení, které není ovlivněno licencí JMeteru, příležitosti k monetizaci jsou tedy výrazně vyšší.

6.2.4 Standalone

Naivní a nejdražší přístup ke gamifikaci JMeteru (potažmo testování) je napsat si ho od nuly. Vznikne tak program, který s JMeterem nesdílí technologie. Zásadní problém je zde ten, že JMeter je vyvíjený už skoro dvacet let. Během té doby se přidala spousta funkcionalit a přestože ne každá musí být replikována, velké množství z nich replikováno být musí, aby nový program neztrácel funkčně na freeware řešení. Vyvinout nový program je velký projekt, který se promítne do vyšší ceny programu. Přesto ale se jedná o poměrně lákavé řešení a můžu ho doporučit software vývojářům, kteří mohou využít síly své značky k propagaci. Menším startupům bych doporučil spíše předchozí řešení založené na upraveném export-import pro snížení nákladů.

6.2.5 Shrnutí

Zvolený přístup závisí ve velké míře na rozhodnutí člověka, kdo se rozhodne implementovat vizualizační prvky do testovacího prostředí. Pokud by se jednalo o lokální projekt pro vlastní účely, např. v softwarehouse týmu, kde se neočekává komerční využití (myšleno: prodej licencí dál) doporučil bych přístup *Plugin*, protože je nejméně náročný na implementaci a má nejlepší poměr ceny vůči vlastnostem.

Pokud by se jednalo o záměr firmy vydat gamifikovaný testovací software pro komerční účely, bylo by zapotřebí více hloubkové analýzy včetně analýzy cen a rozhodnutí, do jaké míry propracovat GUI. V tomto ohledu je velmi těžko radit, jaký přístup zvolit. Obecně platí, že čím složitější vizualizace, tím více samostatný musí program stát. Přesto bych radil vyhnout se výrazným modifikacím jádra JMeteru. I když je tento přístup technicky i legálně možný, nedovedu si představit, že by dával smysl po ekonomické stránce.

Závěr

Ve své diplomové práci jsem se zaměřil na téma testování software a jeho gamifikaci. Zatímco testování software je dlouhé a léty prověřené množství best-practise postupů, gamifikace se teprve „hledá“, a to i v prostředích, které ji jsou bližší, jako je marketing. Gamifikace testování naráží především na nezájem testerů měnit zaběhnuté principy a zvyklosti. Manažeři pak zase mají tendenci překládat „gamifikace“ jako „zdržování“. Fakticky používání gamifikované utility (pokud je naprogramovaná dobře) určitě výkonnost nesníží, spíše ji zvýší, nicméně využívání gamifikace jako motivačního nástroje je teprve v zárodku a neexistuje studie, která by tuto hypotézu potvrdila reprezentativními daty. Přesto jsem se pokusil nastínit několik možných principů, z nichž jsem jeden konkrétní, a to vizualizaci, rozvedl dále. Na základě tohoto principu jsem navrhl možnou implementaci a demonstroval ji na prototypu aplikace.

Řešení jsem prezentoval ve svém pracovním kolektivu, kde celkové vyznění bylo všehovšudy neutrální. Někteří by ocenili jiné grafické téma, někteří vyjádřili obavy z nepřehlednosti při větším počtu testů, s čímž se osobně neztotožňuji. Většina kolegů ale uznala, že uživatelské rozhraní JMeteru by „sneslo“ zlepšení. Rovněž se mi nedostalo vysloveně negativní reakce, která by naznačovala, že se jedná o ztrátu času. Z výše zmíněného usuzuji, že se mi podařilo najít oblast, která má potenciál se rozvíjet.

V brzké době nejspíše uvidíme nárůst počtu gamifikačních prvků za účelem motivování zaměstnanců. Tato diplomová práce demonstruje fakt, že se tato gamifikace nemusí týkat pouze managementu nebo administrativních procesů, ale že se dá zapojit i do ryze pracovních postupů, v tomto případě testování software.

Literatura

[1] Black, R. a kolektiv: *Foundations of Software Testing*. Delmar Cengage Learning, 2012, ISBN 9781408044056.

[2] Grady, R.; Caswell, D.: *Software metrics: establishing a company-wide program*. Prentice-Hall, Inc., 1987, ISBN 0-13-821844-7.

[3] Shaukat, K.: Non Functional Requirement. [online], [cit. 2018-04-03]. Dostupné z: `https://www.slideshare.net/KhushbooShaukat/non-functional-requirement-41586605`

[4] Vaněk, D.: Dimenze kvality FURPS+. [online], [cit. 2018-04-03]. Dostupné z: `http://labe.felk.cvut.cz/~marikr/teaching/Y33TSW_10/FEL-04_doplnek_Dimenze_kvality_FURPS.ppt`

[5] Kočí, P.: Kdo si hraje, neuteče? Co je gamifikace a jak může pokračovat válka o naši pozornost. [online], [cit. 2018-04-03]. Dostupné z: `https://www.lupa.cz/clanky/kdo-si-hraje-neutece-co-je-gamifikace-a-jak-muze-pokracovat-valka-o-nasi-pozornost/`

[6] DAMIAN: How Gamification Transforms Learning. [online], [cit. 2018-04-06]. Dostupné z: `https://www.capytech.com/index.php/2017/04/25/gamification-transforms-learning/`

[7] Werbach, K.; Hunter, D.: *For the Win: How Game Thinking Can Revolutionize Your Business*. Wharton Digital Press, 2012, ISBN 9781613630228. Dostupné z: `https://books.google.cz/books?id=pGm9NVDK3WYC`

[8] Čejka, P.: *Informační systém pro řízení finanční bilance zaměstnanců podniku*. České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

[9] Werbach, K.; Hunter, D.: *The Gamification Toolkit: Dynamics, Mechanics, and Components for the Win.* EBL-Schweitzer, Wharton Digital Press, 2015, ISBN 9781613630686. Dostupné z: https://books.google.cz/books?id=RDAMCAAAQBAJ

[10] Wirtanen, J.: Call of the Wolf. [online], [cit. 2018-04-16]. Dostupné z: http://www.cheatcc.com/extra/wwii/images/wwii_1_0.jpg

[11] goldenguy43: Doom 1 screenshot. [online], [cit. 2018-04-16]. Dostupné z: https://nicoblog.org/wp-content/uploads/2015/09/doom1_screenshot1.jpg

[12] Koliba, M.: *Practical Implementation of Gamification for Scrum.* Czech Technical University in Prague, Faculty of Information Technology, 2017.

[13] Apache Software Foundation: Apache JMeter overview. [online], [cit. 2018-04-11]. Copyright notice. Dostupné z: http://jmeter.apache.org/index.html

[14] Loisel, J.: JMETER RESULT ANALYSIS: THE ULTIMATE GUIDE. [online], [cit. 2018-04-16]. Dostupné z: https://octoperf.com/img/blog/how-to-analyze-jmeter-test-results/jmeter-view-results-tree.png

[15] Herceg, T.: visual studio 2017, .net core a nový formát projektů. [online], [cit. 2018-03-06]. Dostupné z: https://www.dotnetportal.cz/clanek/8506/Visual-Studio-2017-NET-Core-a-novy-format-projektu

[16] Autor anonymní: Nástroj Locmetrics. [online], [cit. 2018-04-11]. Dostupné z: http://www.locmetrics.com/

[17] Přispivatelé Apache wiki: JMX Format. [online], [cit. 2018-04-11]. Dostupné z: https://wiki.apache.org/jmeter/JmxTestPlan

Příloha **A**

Seznam použitých zkratek

GUI Graphical user interface

XML Extensible markup language

API Application Programming Interface

SQL Structured Query Language

JMS Java Messaging Services

SOAP Simple Object Access Protocol

UI User Interface

OS Operační Systém

Příloha B

Obsah přiloženého CD

/
- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl.zip` ..... zdrojové kódy implementace
  - `thesis.zip` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF