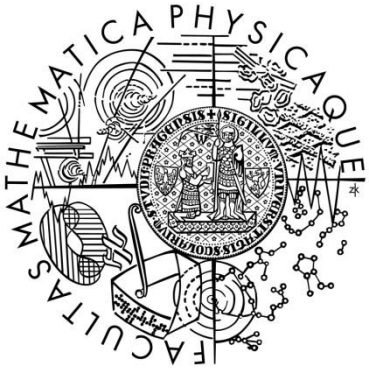
Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Tomáš Hejl

Vývoj webového startupu a přechod na Lean development

Kabinet software a výuky informatiky

Vedoucí diplomové práce: RNDr. Tomáš Holan, Ph.D.

Studijní program: Informatika

Studijní obor: Softwarové systémy

Praha 2012

Na tomto místě bych rád poděkoval RNDr. Tomáši Holanovi, Ph.D. za vedení diplomové práce, za jeho věcné rady, podporu a trpělivost, a také za předcházející vedení projektu Infinity. Dále děkuji Prof. RNDr. Peteru Vojtášovi, DrSc. za rady, přivedení k tématu metodiky Lean Development a za zapůjčení knihy The Lean Startup.

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V …… dne …… podpis

Název práce: Vývoj webového startupu a přechod na Lean development

Autor: Tomáš Hejl

Katedra / Ústav: Kabinet software a výuky informatiky

Vedoucí diplomové práce: RNDr. Tomáš Holan, Ph.D., KSVI MFF UK

Abstrakt: Práce popisuje metodiku Lean Development, sloužící pro řízení vývoje bez známého zadavatele. Dále popisuje již ukončený vývoj konkrétního projektu – webové aplikace, který probíhal bez znalosti této metodiky. Práce pak porovnává, v kterých aspektech by se vývoj tohoto projektu změnil, kdyby byla metodika Lean Development použita, a navrhuje konkrétní řešení pro další postup již v souladu s touto metodikou. Na závěr na příkladu tohoto projektu ukazuje seznam aktivit, které musí vývojářský tým při vývoji bez zadavatele podstoupit.

Klíčová slova: web, startup, lean development

Title: Creating web startup and introducing the Lean development

Author: Tomáš Hejl

Department: Department of Software and Computer Science Education

Supervisor: RNDr. Tomáš Holan, Ph.D., KSVI MFF UK

Abstract: This work describes The Lean Development methodology that can be used for managing development without known customer. Moreover, it describes the development of an already finished project – a web application that did not involve this methodology. After that this work finds out which aspects of this development would change, if the Lean Development methods were integrated, and proposes a specific solution for subsequent development, compliant with this methodology. At the end, this work presents the list of activities that the development team must pass when developing without known customer.

Keywords: web, startup, lean development

Obsah

1 Úvod

1.1 Téma práce

Tématem této práce je metodika Lean Development pro vývoj startupů se zaměřením na webové aplikace a nasazené této metodiky na konkrétním projektu.

Práce popisuje nejprve obecně metodiku Lean Development a její výhody pro vývoj aplikací bez zadavatele.

V dalších částech práce popisuje všechna stádia vývoje projektu Infinity a zaměřuje se zejména na prvky metodiky Lean Development, které tým využil, a které mohl dále zapojit.

Během popisu vývoje projektu je kladen důraz na moderní technologie, aktuální pro rok 2012, a jejich současnou i budoucí použitelnost.

Poslední práce tvoří návrh dalšího postup pro projekt Infinity, již zcela podle metodiky Lean Development.

Závěr obsahuje stručný seznam všech aktivit a postupů, které musí autor softwarového díla bez zadavatele znát, aby mohl ve světě startupů uspět.

1.2 Co je to startup

Startup je firma, založená s účelem vytvořit nový produkt nebo službu za podmínek extrémní nejistoty. Tuto definici nabízí Eric Ries (1), který jako jeden z prvních popsal metodiku Lean Development jako celek.

Za startup se tedy může považovat jakákoli firma, která je založena za účelem vzniku zcela nového produktu, aniž by předem měla tento produkt přesně specifikován, aniž by měla předem zařízený odbyt. Firma tak postupně vytváří a mění produkt, o němž není jisté, zda vůbec své uplatnění najde. Startup tak musí postupně hledat své uživatele, zákazníky či klienty, a postupně upravovat svůj produkt tak, aby na trhu uspěl.

1.3 Projekt Infinity

Infinity je softwarový projekt, vypracovaný a obhájený na Matematicko-fyzikální fakultě Univerzity Karlovy v Praze v letech 2011 až 2012. Autor této práce byl jedním ze čtyř programátorů v autorském týmu projektu a významnou měrou se podílel na vzniku projektu.

V rámci tohoto projektu vznikla webová aplikace – internetová hra, běžící v běžném prohlížeči. Při vývoji byly využity moderní technologie, z nichž některé budou zmíněny v dalších částech této práce. Vývoj Infinity probíhal podle pravidel Agilního vývoje. Tým v době vývoje neznal metodiku Lean Development, přesto postupně zaváděl vlastní úpravy agilní metodiky, které se metodice Lean Development podobaly.

Projekt Infinity bude blíže popsán v dalších kapitolách, přičemž na jeho vývoji budou ilustrovány vlastnosti metodiky Lean Development.

2 Lean Development - Motivace

Hlavním problémem vývoje bez zadavatele je – jak již napovídá samotná definice startupu – nejistota. Při vývoji se zadavatelem je předem jistý odbyt pro vznikající software, a také je předem dobře známa jeho cílová skupina, ne-li přímo konkrétní uživatelé. Dobře specifikované je i prostředí, na kterém software poběží a další informace o jeho budoucím využití.

Při programování startupů tyto informace chybí, postupně vzniká produkt, který si teprve bude své využití hledat, stejně jako své uživatele. Hrozí tedy, že kvůli mnoha faktorům své využití nenajde a uživatelé nebudou mít zájem produkt využívat, natož za něj platit.

Tyto problémy řeší relativně nová metodika „Lean Development“, která obdobně jako její předchůdkyně (Waterfall, Agilní vývoj ad.) pochází z oblasti průmyslové výroby.

2.1 Původ v průmyslu

S přístupem „Lean Manufacturing“, ze kterého se postupně vyvinula metodika „Lean Development“, přišla poprvé společnost Toyota (1 str. 18). Toyota ale stavěla na ještě starších základech – principy jako JIT (Just-in-time production) a další jednotlivé metody již existovaly, inženýr Taiiči Óno je pouze využil, propojil dohromady a vytvořil systém (TPS, Toyota Production System), který Toyota využívá dodnes a jehož zobecněním vznikla metodika Lean Manufacturing (2).

Tento přístup řeší podobné problémy, jako Lean Development na poli vývoje softwaru, ale základní princip je zcela stejný – minimalizace odpadu. Jak a proč se tato myšlenka projevuje v programování, bude zmíněno později.

V Toyotě tento přístup například minimalizuje zbytečné náklady na výrobu a skladování – podle principu Just-in-time production tam, kde jiné firmy vyrobí do zásoby velké množství součástek (protože hromadná výroba je levnější), Toyota vyrobí dílů přesně tolik, kolik jich aktuálně potřebuje. Cena za jeden kus je mírně vyšší, ale úspory se objeví jinde – není třeba tolik skladovacích a transportních kapacit, a hlavně, pokud dojde k chybě (součástka nefunguje, součástka je nahrazena novou verzí, produkt přestal být vyráběn a tak

už tyto součástky nebudou nikdy potřeba…), dojde k menším škodám, zatímco firmy vyrábějící tradičním způsobem musí likvidovat větší množství zbytečného odpadu.

Přístup Lean Manufacturing také určuje postupy, které tyto škody dále snižují – snaží se co nejdříve zjistit, jak je možné součástku vylepšit (a jestli vůbec má její vylepšení smysl) a obecně zajistit, aby existovaly jen ty součástky, které jsou nezbytně potřeba, a aby byly co nejlepší co nejdříve, ne po dlouhých letech, během kterých by se zbytečně vyráběla jejich horší varianta.

Další součásti systému Toyota Production System zahrnují dále i sociální pokyny typu „Respekt k lidem“ nebo prvky motivující k dodržování dlouhodobé filozofie projektu či neustálým inovacím.

Adaptací některých z těchto postupů do oblasti vývoje softwaru vznikla metodika Lean Development.

2.2 Lean Development – vývoj softwaru

Cíle metodiky Lean Development na poli vývoje softwaru jsou naprosto stejné jako cíle původních principů Lean Manufacturing od Toyoty. Snaží se minimalizovat odpad, tedy v případě programování zbytečnou práci – programování součástí, které nezlepší přijetí finálního produktu, či mu dokonce uškodí, a tedy ve výsledku nepřispívají k zisku vznikající firmy – startupu. V celkovém pohledu Lean Development brání vzniku softwaru, který by nenašel své využití či softwaru, který svým uživatelům nenabídne takovou hodnotu, aby se firmě jeho vývoj vyplatil.

Pro odstraňování „odpadu“ z vývoje softwarového startupu využívá metodika Lean Development tři hlavní pilíře:

- **Validated learning** (Ověřené učení)
- **Smyčka Build-Measure-Learn** (Vytvoř-Změř-Pouč se)
- **Innovation accounting** (Inovační metriky)

2.2.1 Validated learning

Ve většině zavedených firem jsou jednotlivá oddělení striktně odloučena – programátoři jako jeden či více týmů vytvoří produkt, jiné oddělení zajistí propagaci, jiné oddělení zajistí prodej jako takový. Další oddělení pak zpracovává případné připomínky uživatelů produktu, a nějakým způsobem je předává opět programátorům, kteří tyto návrhy či stížnosti mohou promítnout do vývoje další verze softwaru.

Důvod je zřejmý – odborníci například na propagaci nemůžou být ve velké firmě přiděleni jen jednomu projektu, ve kterém by většinu času jen čekali, než programátoři dodají novou verzi. Místo toho jsou jejich schopnosti využity jinde, a s žádným projektem tak nejsou dostatečně propojeni.

Pro potřeby startupů je toto naprosto nevyhovující přístup – startup musí rychle reagovat na sebemenší reakce uživatelů, a proto je nezbytné těsnější propojení jednotlivých odborníků – shánění nových uživatelů a zapracovávání jejich odezvy musí probíhat kontinuálně, jinak startup nemůže včas zjistit, že se vydal špatným směrem.

S tímto korporátním přístupem se pojí i několik dalších faktů, které v prostředí vývoje bez zadavatele neobstojí.

Pro programátory v takové společnosti je hlavním cílem naprogramovat zadaný produkt včas, a nepřekročit přitom požadované náklady. Na poli startupů toto samozřejmě neplatí – je k ničemu, že jsme vytvořili produkt za veškeré peníze, pokud produkt nikdo nechce.

Také v současnosti převážně platí, že programátor by měl většinu času programovat, časté porady podle nadřízených spíše kazí produktivitu. Ve startupu je opět lepší více komunikovat a jednat, než celý den programovat – co když celý den programuji něco, co se ve výsledku stane zbytečným odpadem?

Je nutné si uvědomit, že při vývoji startupu má učení velkou cenu. Je velmi důležité co nejdříve zjistit, co zákazník našeho produktu potřebuje, čím ho produkt zaujme, co je třeba, aby produkt uspěl na trhu. Bez těchto zjištění ani stoprocentní efektivita nepostačí k vytvoření úspěšného softwaru.

Problémem učení je to, že se jeho výsledky nedají nijak změřit. Nikde nevidíme hmatatelné výsledky, žádná čísla, o kterých bychom mohli říct (natož precizně spočítat), že se nám učení opravdu vyplatilo. Přesto je však nezbytnou součástí vývoje startupu.

2.2.2 Build–Measure–Learn

Doslovný překlad tohoto principu mluví sám za sebe – po naprogramování nové části či vlastnosti produktu je třeba změřit efekt této změny a z výsledku měření se poučit a příslušně na něj reagovat.

Reakcí může být buď vytrvání v současném směru (anglicky Persevere), například přidávání dalších podobných změn či vylepšování současných principů, zatímco druhá možná reakce, anglicky nazývaná Pivot, značí změnu směru vývoje produktu – pokud nová změna fatálně neuspěla, může být na místě prohlášení této konkrétní cesty za slepou uličku (tato změna může být například úplně odstraněna), nebo dokonce může být opuštěn některý z fundamentálních principů produktu a ten se tak může naprosto změnit.

Tato rozhodnutí o „změně směru“ bývají nejtěžší částí vývoje produktu – někdy totiž znamenají opuštění velkého množství kódu a absolutní změnu přístupu, a to není rozhodnutí, které by jakýkoli vývojářský tým dělal rád. Tento krok je však často nezbytný, a nedostatek odvahy ke změně navzdory výsledkům měření může vést k neúspěchu celého produktu.

Metodika Lean Development také silně doporučuje co největší zkrácení jednotlivých cyklů smyčky Build-Measure-Learn.

U velkých výrobců softwaru je často zvykem vydávat novou verzi produktu v pravidelných a dlouhých cyklech, například jednou ročně. Takový přístup se vůbec neshoduje s principy smyčky Build-Measure-Learn a to hned z několika důvodů.

Při dlouhém vývojovém cyklu se do nové verze produktu dostane často velké množství nesouvisejících změn. Je pak velmi těžké, čí přímo nemožné, změřit dopad těchto změn jednotlivě. Pokud nová verze neuspěla, může za to více změn, nebo jen jedna konkrétní, kterou bychom mohli snadno odstranit? Pokud naopak uspěla, nestačilo by na stejný úspěch menší množství změn, nejsou některé zbytečné? V těchto dlouhých cyklech není možné efektivně odstranit „odpad“, protože neexistuje způsob, jak ho ve velkém množství novinek identifikovat.

I kdyby bylo možné „nevýhodnou“ novinku přesně určit, mohla se od té doby promítnout do mnoha míst v produktu, stejně tak je možné, že její původní tvůrce v době vydání této verze už ve firmě nepracuje, a tak je oprava často velmi obtížná.

Při tomto systému „velkých dávek“ hrozí také takzvaná „**spirála smrti velké dávky**“ (anglicky Large-Batch spiral of death). Při velkém množství změn, které se mají vypustit společně, hrozí, že se objeví velké množství větších či menších chyb. Při jejich opravování se odkládá datum vypuštění této hromadné změny, a často se stihne přidat další množství nových změn, a další chyby. Pak je velmi reálné, že taková dávka nebude vypuštěna nikdy, ze strachu před narůstajícím množstvím chyb, a produkt bude ukončen definitivně, přesto že na něm bylo provedeno velké množství práce, která se nyní ukázala jako naprosto zbytečná.

2.2.3 Innovation accounting

Předchozí pilíře metodiky Lean Development často zmiňují testování, měření, učení se z výsledků. Je tedy nutné stanovit metriky, které jsou při měření dopadů změn produktu použity.

U softwaru, vyvíjeného pro zadavatele, bývá hlavním znamením úspěchu velký počet uživatelů, nebo i jen růst této veličiny. Čím více lidí můj software používá (příp. ho zakoupilo), tím lepší produkt (včetně jeho propagace atd.) byl.

Mezi startupy tato metrika opět selhává – počet uživatelů stabilně roste prakticky u všech startupů, sehnat další a další zákazníky většinou není velký problém. S růstem uživatelské základny se ale úspěch startupu nepojí – pokud se například jedná o zákazníky, kteří produkt pouze zkusili, a již nikdy se k němu nevrátí, nebo pokud využívají ty části produktu, které firmě nepřinášejí žádný zisk. Klasickým případem jsou například neplacené varianty produktu, které uživatelům postačují, placenou verzi si mezitím prakticky nikdo nekoupí.

Nepomáhá ani další oblíbená technika, využívaná při programování podle agilního či vodopádového přístupu, milníky práce („milestones“). U softwaru se zadavatelem je většinou určeno, kolik by zhruba měl mít uživatelů, aby byl úspěšný – pokud například software pro automobilky používá 50% všech automobilek, je jistě úspěšný. U startupů, kde není předem známá často ani cílová skupina, natož přímo cíloví zákazníci, je prakticky

nemožné odhadnout, kolik by v případě úspěchu měl mít uživatelů, i kdyby tato metrika vůbec měla smysl.

Je tedy nutné stanovit jiné, nové metriky, které lépe odpovídají tomu, co od uživatelů startupu požadujeme. Často je nezbytné přizpůsobit je na míru konkrétnímu projektu, generické metriky (jako zmiňovaný celkový počet uživatelů) většinou nestačí, resp. nefungují.

Detailům inovačních metrik se věnuje kapitola 4.2 Metriky.

3 Lean Development – Vize a strategie

3.1 Vize a strategie

Eric Ries v knize Lean Startup dělí vývoj produktu do tří vrstev (1 str. 22). Těmito vrstvami jsou Vize, Strategie a samotný Produkt. Vize se po dobu vývoje nemění. Strategie, jakožto souhrn konkrétních principů, funkčností produktu atp. se může občas změnit (jedná se o již zmíněný Pivot, kterému se dále věnuje kapitola 4.3 Innovation accounting), a produkt jako takový se samozřejmě mění často, jak postupně vzniká a přibývají další jeho funkce.

3.1.1 Vize

Každý projekt začíná vizí. **Vize** je souhrn základních nápadů a myšlenek, pilířů právě vznikajícího produktu, jeho hlavních vlastností, které produkt definují a po dobu vývoje se nebudou měnit.

Strategie určuje konkrétní postupy při vývoji produktu a také konkrétní směr, kterým se produkt bude ubírat – například využité technologie nebo jednotlivé funkce produktu.

3.1.2 Strategie

Eric Ries v knize Lean Startup doporučuje postavit strategii projektu na dvou hlavních hypotézách – hypotéze hodnoty (**value hypothesis**) a hypotéze růstu (**growth hypothesis**). Tyto hypotézy považuje za nejdůležitější předpoklady, na nichž závisí celý vývoj (a případný úspěch) produktu.

Value hypothesis

Je nezbytné vědět, zda výsledný produkt přinese uživatelům nějakou hodnotu, něco, co je přiměje produkt využívat. Existují vůbec uživatelé, kteří chtějí nebo potřebují náš produkt? Existuje problém, který náš produkt řeší?

Tato hodnotová hypotéza tedy stanovuje, jakým způsobem přinese produkt svým uživatelům nějakou hodnotu.

Druhou důležitou částí hodnotové hypotézy je také otázka „Jaké hodnoty přinese tento produkt firmě?“. Typicky je třeba určit, jakým způsobem bude aplikace přinášet zisk startupu. Nemusí se vždy jednat o finanční zisk (což je ale nejčastější případ), může se jednat

i o důležitá data nebo zákaznickou základnu, kterou později startup využije jinak (např. pro další produkt).

Tyto otázky musí být zodpovězeny co nejdříve, aby se předešlo vývoji produktu, který své uplatnění nenajde či nepřinese firmě žádný zisk. K jejich zodpovězení se používá nejčastěji tzv. Minimální produkt, více o něm viz stejnojmenná kapitola 4.1.

Growth hypothesis

Eric Ries rozlišuje několik různých způsobů, jakými může aplikace růst a získávat nové zákazníky.

„**Sticky Engine of Growth**“ spoléhá na dlouhodobé zákazníky, kteří zůstávají, a firma na nich vydělává opakovaně. Příkladem mohou být například výrobci antivirových programů – většina uživatelů zůstává u svého vybraného produktu, a kupuje nové verze, nové aktualizace.

„**Viral Engine of Growth**“ využívá propagace produktu přes samotné zákazníky – ať už prostřednictvím sociálních sítí, nebo třeba jen ústní metodou. Tento způsob využívá například většina her na platformě aplikace Facebook – hráči jsou ve hře vyzýváni k rozesílání pozvánek mezi své přátele. Další uživatele přivádějí ti stávající také přímou (např. ústní) propagací.

Poslední možností je „**Paid Engine of Growth**“, kde nového zákazníka získají placené postupy – klasický marketing, reklamy atp.

Produkt musí mít ve své strategii jasně určeno, který z těchto tří způsobů volí – na tomto rozhodnutí pak stojí další části produktu, a pokud během vývoje dojde k tak závažné změně strategie, že se změní i tato klíčová hypotéza, často to znamená velké ztráty, resp. velké množství „odpadu“.

Typickým případem může být zvolení placeného marketingu pro cílovou skupinu, která na něj téměř nereaguje a lepších výsledků by dosáhlo virální šíření – peníze na reklamy pak byly využity zbytečně. Naopak, odpadem může být i programování sociálních funkcí pro cílovou skupinu, která odmítá produkt mezi své přátele šířit, a naopak by lépe reagovala na reklamy klasického placeného marketingu.

Změny strategie

Strategie se mění jen v případě, že se stávající volba ukáže jako nevhodná – měření například ukáže, že některý z předpokladů neplatí, a je třeba produkt předělat tak, aby lépe odpovídal zjištěným hodnotám.

Příkladem takové změny může být například přechod z webové aplikace na klasický software – taková změna by musela být vynucena zjištěním závažných nedostatků ve fungování stávající strategie, a je zřejmé, že by měla rozsáhlé následky, včetně např. změny veškerých technologií, na kterých původní webová aplikace stála.

4 Lean Development – Principy

4.1 Minimální produkt

4.1.1 Co je minimální produkt

Jednou ze základních myšlenek metodiky Lean Development je takzvaný Minimální produkt (v originále **Minimum Viable Product**, doslova minimální životaschopný produkt). Jedná se o nejjednodušší způsob, jak co nejdříve ověřit všechny základní hypotézy projektu (hypotézy viz kapitola 3.1.2), ale i všechny dílčí hypotézy v pozdějším vývoji (typicky „je vhodné přidat tuto novou funkci do produktu?“).

Princip minimálního produktu říká, že je třeba každou hypotézu ověřit přímo „u zdroje“, tedy u zákazníků/uživatelů. Doporučuje tedy vytvořit jakoukoli „demoverzi“ testovaného produktu nebo vlastnosti či funkčnosti, a otestovat ji přímo na lidech z cílové skupiny.

Nemusí se jednat jen o betaverzi samotného programu, jak tomu často bývá – minimální produkt může být mnohem jednodušší. Může se jednat jen o nedokončený prototyp, kterému chybí většina funkcí a stávající funkce nefungují 100% správně – to většinou nevadí, protože cílem minimálního produktu je otestovat, zda produkt jako takový může uspět. Pokud uživatel nepochopí či odmítne používat tento jednoduchý prototyp, stejné reakce se nejspíš dočká i později produkt samotný.

Podle Erica Riese (1 str. 64), resp. jeho rozhovorů s podnikateli (např. Mark Cook z Kodak Gallery) pomáhá minimální produkt zodpovědět základní otázky vývoje bez zadavatele:

- Vnímají vůbec uživatelé existenci problému, který náš produkt má řešit?
- Pokud existuje řešení tohoto problému, zaplatí za něj?
- Zaplatí za něj nám?
- Dokážeme vytvořit řešení?

Zajímavé je obzvláště srovnání s vývojem programu se zadavatelem – tam se hledá odpověď pouze na otázku poslední, všechny ostatní jsou předem zodpovězeny – zadavatel zjevně vnímá existenci problému a je ochoten nám za jeho řešení zaplatit.

Na poli startupů musí nejprve tým najít odpovědi na tyto otázky, a to co nejrychleji – pokud se po dlouhé době vývoje zjistí, že odpověď na libovolnou z otázek je negativní, probíhal celý vývoj zbytečně. Právě v tom pomáhá minimální produkt – nabídne jednoduchou testovací verzi, jejímž užíváním uživatelé sami odpovědí na první otázku, a rozhovory s těmito uživateli (nebo vypuštění dalšího minimálního produktu s placenými funkcemi) pak pomohou zodpovědět zbylé dvě otázky.

Testovacím produktem nemusí být dokonce ani software, připomínající výsledný produkt – může se jednat jen o experimentální prototyp, obsahující například jen jednu vybranou funkčnost. Jednoduchost není na škodu, naopak, je přímo doporučena – pomůže zodpovědět otázky konkrétněji. Pokud prototyp uspěje, potenciální zákazníci mají zájem o tuto konkrétní funkčnost, a ta se tak nestane odpadem, ale může začít její zapojování do finálního produktu. U komplexnějšího prototypu je obtížnější zjistit, která z jeho částí způsobila jeho úspěch či neúspěch.

Minimálním produktem nemusí být dokonce vůbec žádný software. Například známý startup Dropbox, nabízející řešení k automatickému sdílení souborů mezi různými počítači, začal ještě jednodušším způsobem (1 str. 98). Vytvořil krátké video, které názorně ukazovalo, jak by výsledný produkt mohl fungovat. V té době ještě žádný produkt neexistoval, byla jen dokončena základní analýza. Video se setkalo s velkým úspěchem, odezva od potenciálních zákazníků jasně ukázala, že o takový produkt mají zájem, že řeší problém o kterém takřka ani netušili, že ho mají, a že za toto řešení budou ochotni zaplatit. Tým tak mohl začít programovat bez strachu, že vyvíjí zbytečný software, který nikdo nebude používat.

Podobně mohou jako minimální produkt sloužit například náčrty finálního produktu (například v podobě wireframe, viz kapitola 6.7 Nástroje pro analýzu a design) či funkční prototypy uživatelského rozhraní (které ale neposkytují žádnou skutečnou funkcionalitu), ze kterých uživatel pochopí, jak bude výsledný produkt fungovat.

4.1.2 Problémy s Minimálním produktem

S principem minimálního produktu je spojeno několik základních problémů, se kterými musí vývojáři pracující podle metodiky Lean Development počítat.

Nekvalita minimálního produktu

O minimálním produktu je jasně řečeno, že nemusí být příliš kvalitní – celková kvalita u něj není v žádném případě cílem či nutností. Problém nastává v případě, kdy zákazníci chápou kvalitu jinak, než vývojářský tým. V této fázi je tedy nutné s „testovacími subjekty“ neboli pokusnými uživateli intenzivně komunikovat, a zjistit přesně, co jim na prototypu vadí. Pokud se jedná o kvalitativní nedostatek prototypu, je možné chybu v této fázi vývoje zcela přejít. V nejhorším případě ale taková chyba prototypu může zkreslit celkové výsledky experimentu – zákazníky může odradit od testování samotné klíčové funkcionality, na kterou je prototyp zaměřen – a proto je vždy nutné o ní vědět.

Výhodou nasazení nekvalitního prototypu je často zjištění, které typy chyb zákazníkům vadí, a které naopak vnímají jako nepodstatné. Toto zjištění usnadní další vývoj a nasazování produktu – tým už ví, že na některý druh problémů se může zaměřit později, a jiný musí naopak řešit prioritně, protože ho zákazníci vnímají jako důležitou a nepříjemnou chybu.

Dokonce se může stát, že to, co tým vidí jako nedodělek, jasný znak nekvality, mohou zákazníci vnímat jako pravý opak. Jedná se většinou o dočasná řešení, která v prototypu nahrazují plánované komplexnější metody – a testování s uživateli již v této fázi ukáže, že náročné a drahé komplexní řešení není vůbec potřeba, že jsou spokojeni s onou dočasnou a podle názoru týmu „nekvalitní“ variantou. To je další případ, kdy testování s minimálním produktem zabrání vzniku odpadu, tedy zbytečné práce – náročnější varianta sice mohla uspět stejně či dokonce více než tato dočasná, ale není to jisté, zatímco jednoduchá varianta je již v tuto chvíli hotová a u uživatelů oblíbená, není tak nutné riskovat a tvořit nové a drahé řešení.

Právní problémy minimálního produktu

Dalším problémem minimálního produktu jsou často obavy z krádeže nápadu – prototyp většinou není výhodné patentovat či jinak zákonem chránit, a tak se tým obává, že konkurence využije jeho rozpracované nápady a postaví na nich vlastní produkt.

Tyto obavy jsou ale většinou liché – prototyp většinou bývá malý a testovaný jen malou skupinou lidí, konkurence si tak takového testování nemusí ani všimnout, zvlášť když se jedná o projekt neznámého začínajícího startupu. Ve chvíli, kdy probíhá tak rozsáhlé

testování, že ho konkurence zaznamená, je většinou vývoj již v pokročilé fázi, a navíc má startup právě díky rozsáhlému testování k dispozici velké množství zkušeností a dat, díky kterým má nad konkurencí klíčovou výhodu – díky nim dokáže vytvořit výsledný produkt rychleji a/nebo lépe než konkurence.

Časový problém minimálního produktu

Posledním a největším problémem je otálení s vypuštěním minimálního produktu, ať už z obavy z nekvality, či například ze strachu vypustit prototyp, který nereprezentuje kompletní vizi plánovaného produktu. Rizika jsou zřejmá – tým nemá šanci bez minimálního produktu zjistit, zda jsou jeho základní hypotézy pravdivé, zda jsou kladné odpovědi na všechny otázky z předchozí kapitoly 4.1.1. Tým pak ztratí všechny výhody metodiky Lean Development – a navzdory snaze vytvoří produkt, který nenajde své uplatnění.

4.2 Metriky

V projektech se zadavatelem bývá znamením úspěchu kladný zisk a růst počtu uživatelů. U startupů tomu tak být nemusí, toto kritérium splňuje většina z nich, i ty, co nakonec selžou. Stejně tak nepomáhá stanovit milníky vývoje, a po jejich uplynutí zjistit, zda projekt naplnil očekávání – většinou nemáme žádný způsob, jak předem efektivně odhadovat cílová čísla.

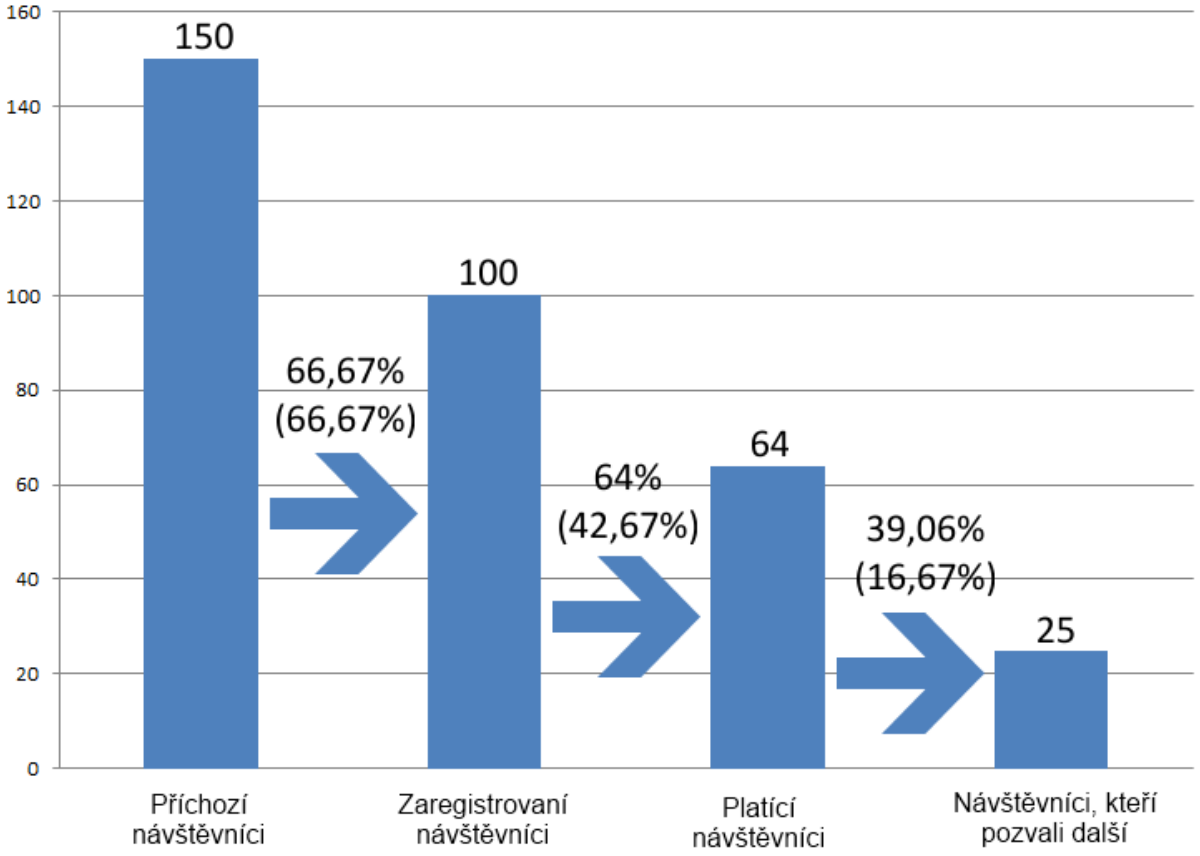
Je tedy nutné zavést nové metriky, nové způsoby měření a odhadování úspěšnosti.

Staré a jednoduché metriky, které pro startupy nejsou příliš vhodné, označuje metodika Lean Development jako „vanity metrics“, tedy v překladu „marnivé metriky“.

4.2.1 Funnel analysis

Funnel analysis, tedy takzvaná trychtýřová analýza, se zabývá uživatelskými akcemi, které vývojáře aplikace nejvíce zajímají. Tyto aktivity se v jednotlivých projektech liší, každý tým musí najít ty, které jsou důležité právě pro jeho obor a jeho typ produktu. Analýza pak sestavuje „trychtýř“ aktivit – kolik procent uživatelů provedlo první aktivitu? Kolik z nich se zapojilo víc, a došlo ke druhé? Kolik z nich ke třetí?

Typický případ funnel analysis ilustruje následující graf:



Tento příklad ukazuje analýzu typickou pro webovou aplikaci, která sází na „viral engine of growth“, tedy na propagaci prostřednictvím virálního šíření. Sleduje celkový počet návštěvníků aplikace, který ale pro startup není vůbec zajímavý. Zajímavá jsou teprve procenta uživatelů, kteří přináší začínající firmě zisk.

Registrovaní zákazníci přináší minimálně svá data a je zajímavé sledovat, kolik procent z celkového počtu uživatelů registraci podstoupí. Pokud je procento nízké, je nutné se ptát – je registrace složitá či nepřístupná? Nemají uživatelé zájem aplikaci používat? Nebo pouze nemají zájem o registraci, protože jim stačí ta část aplikace, která registraci nevyžaduje? Odpovědi na tyto otázky (získané pomocí dotazování uživatelů či experimentálně) mohou žádané procento registrací výrazně zvýšit – nejprve je ale nutné vědět, že tento problém existuje. Proto je tento typ analýzy výhodný.

Další vrstvou uživatelů jsou ti registrovaní uživatelé, ze kterých plyne finanční zisk – může se jednat například o uživatele placené verze či jen dobrovolné přispěvatele. Tak či tak je pro začínající firmu klíčové, aby toto procento bylo co nejvyšší.

Pro startup, který sází na virální šíření, je pak nezbytná i poslední část grafu, tedy počet platících návštěvníků, kteří mezi platící skupinu pozvou další uživatele a tedy získávají pro aplikaci další zdroj financí.

Skutečnou cenu tato analýza získává až s přidáním třetího rozměru, konkrétně času. Není tolik zajímavé, kolik procent lidí se zaregistrovalo, ale jak se toto procento změnilo například po úpravě registračního formuláře. Pokud procentu zaregistrovaných vzrostlo, změna byla úspěšná, v opačném případě se jednalo o zbytečnou práci a problém byl zřejmě jinde a je nutné ho identifikovat.

Problémem ale zůstává fakt, že celková čísla jsou neustále ovlivněna celou minulostí produktu a nemají takovou vypovídající hodnotu, jak by bylo potřeba. Proto vznikla analýza, o které mluví následující kapitola.

4.2.2 Cohort analysis

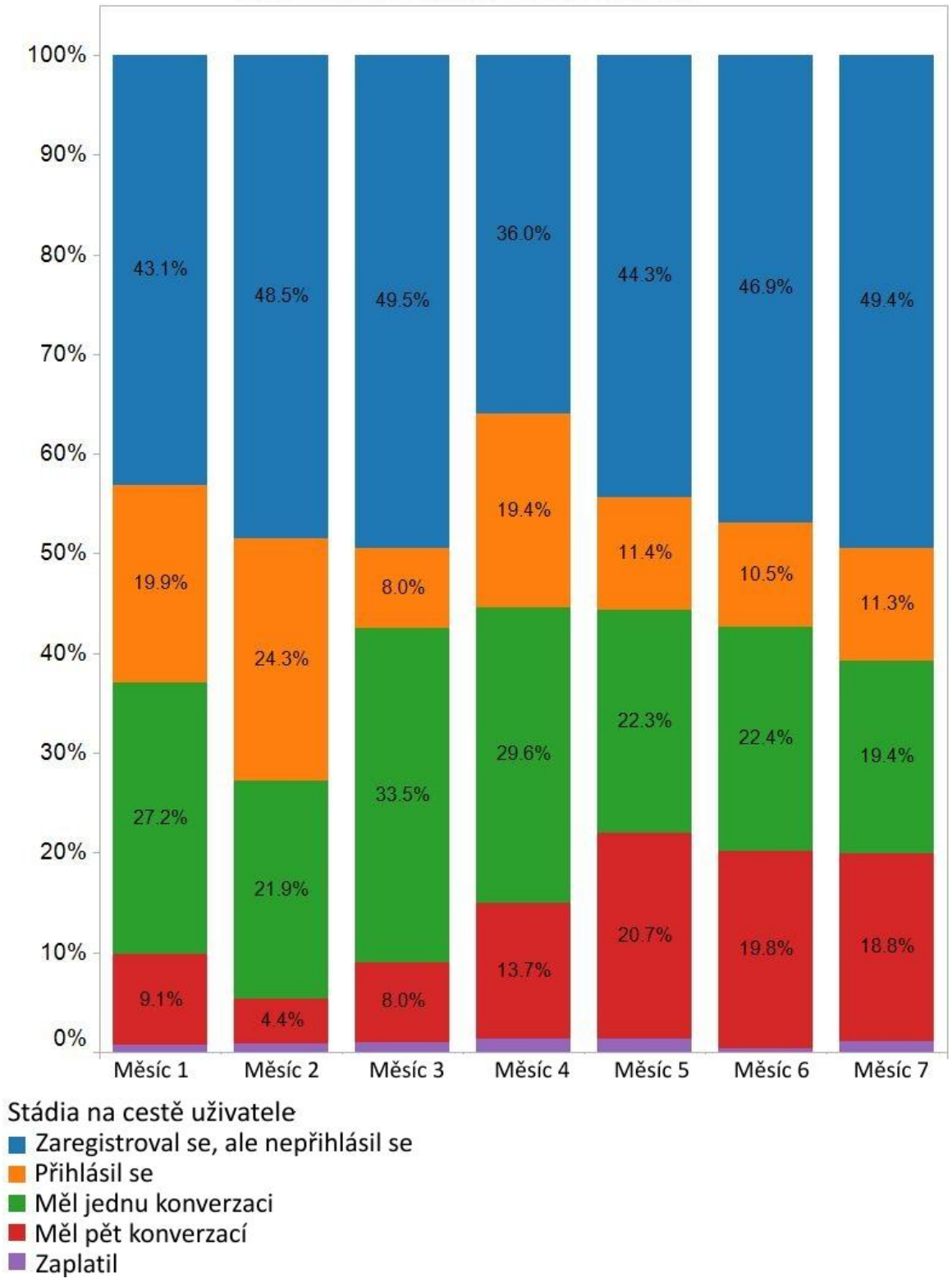
Cohort analysis přidává trychtýřové analýza pro určité časové období další význam. Neměří totiž výsledky pro všechny uživatele produktu, ale zabývá se jen určitou předem definovanou skupinou uživatelů. Touto skupinou typicky bývají uživatelé, kteří poprvé aplikaci navštívili v zadaném období, například v konkrétním měsíci. Místo abychom měřili celková procenta pro daný měsíc, zajímáme se tento měsíc jen o zcela nové uživatele.

Zbavíme se tak ovlivněných dat – například uživatelé, kteří poprvé přišli před mnoha měsíci, a viděli starý a nefunkční registrační formulář (a proto ho nevyužili), už jsou ovlivněni, a i pokud se vrátí, je malá šance, že se o registraci pokusí znovu. Pokud ale mám k dispozici informaci, že už tu byli (například rozpoznání podle IP adresy či cookies), jednoduše je do aktuální kohorty-skupiny nezařadím a zajímám se jen o opravdu nové uživatele.

Přínos je zřejmý – trychtýřová analýza kohorty pro daný měsíc měří pouze chování uživatelů, které ovlivnil stav aplikace v měřeném měsíci, nic jiného. Můžeme tak lépe sledovat, jak se vývoj aplikace projevil – například opravený registrační formulář by se v celkových číslech příliš neprojevil – většina uživatelů by opět očekávala starou nefunkční

verzi a proto by celkové procento registrací příliš nevzrostlo. Procento registrací zcela nových uživatelů, kteří o bývalé chybě nevěděli, by však vzrostlo výrazněji.

Ukázka grafu Cohort analysis:



Tento ilustrační obrázek ukazuje příklad, uváděný Ericem Riesem (1 str. 122) – jedná se o graf aplikace, umožňující ve virtuálním světě konverzace mezi uživateli. Tým zjevně zajímalo nejen to, kolik lidí se zaregistrovalo, ale kolik lidí se po registraci vůbec vrátilo a přihlásilo. Graf ukazuje, že tým během těchto měsíců vyřešil problém s nedostatkem uživatelů, kteří vůbec zkusili konverzaci prostřednictvím této aplikace, ke konci období již bylo relativně vysoké procento i těch uživatelů, kteří konverzovali víckrát. Přesto však zůstávalo nízké procento uživatelů, kteří za službu zaplatili, a proto se v následujícím období zaměřili právě na tento problém.

Analýza tedy pomáhá identifikovat problémy v aktuální podobě produktu – jasně vidíme, jak se chování nových uživatelů změnilo oproti minulým měsícům, a jelikož známe historii projektu, dokážeme odhadnout, která změna aplikace jejich chování způsobila, a na jaké cíle se musí zaměřit budoucí úpravy.

4.2.3 Split testování

Vhodným doplňkem zmiňovaných metrik je takzvané split testování, známé také jako A/B testování. Pomáhá ještě přesněji určit, jak nová změna produktu ovlivňuje chování uživatelů.

Princip tohoto testování je jednoduchý – sledovaní uživatelé se rozdělí na dvě skupiny, a každá bude používat jinou verzi produktu. Může se jednat například o produkt bez nové změny a produkt již tuto změnu obsahující, či například produkty se dvěma verzemi stejné změny.

Sledování dostatečně velkého počtu uživatelů pak snadno odhalí, která z obou skupin ukázala lepší výsledky ve sledovaných veličinách, a tedy která z obou verzí produktu je lepším základem pro další pokračování vývoje.

4.2.4 Metriky 3A

Podle Erica Riese (1 str. 143) by měly metriky, využívané k měření úspěšnosti startupů, splňovat tzv. „3A“. Do češtiny by se tento princip mohl přeložit jako „3D“ (dokazatelnost, dostupnost, dohledatelnost).

Actionable Metrics

Metriky, které E. Ries nazývá „actionable“ (do češtiny by se dalo přeložit jako „dokazatelné“), dovolují vždy určit příčinu a její důsledek. Tyto metriky pak dovolují určit, která změna aplikace měla kladný a která záporný vliv, a případně tento vliv konkrétněji vyčíslit.

Typickým příkladem je výsledek cohort analysis – můžeme vždy říci, že výsledek této analýzy závisí pouze na stavu aplikace v měřeném období a není ovlivněn staršími akcemi.

Důležité je také při testování minimalizovat testované časové období, a testovat vždy co nejmenší počet změn – při větším počtu změn najednou jen těžko odhadneme, která z nich způsobila pozorovanou změnu výsledků.

Accessible Metrics

Metriky musí být přístupné, čitelné, srozumitelné.

Výsledky metrik musí sloužit svému účelu – učení. Z výsledku analýzy se musí často poučit všichni členové týmu – a příslušně pozměnit aplikaci samotnou, komunikaci s uživateli nebo třeba přístup k marketingu a propagaci produktu. Je tedy nutné, aby takovou metriku dokázali pochopit všichni, ne jen člen týmu, který analýzu vytvořil.

Typickým příkladem je například použití jasných a srozumitelných jednotek – „celkový počet uživatelů“ pochopí jistě každý, zatímco spojení „aktivní uživatelé“ může pro každého člena týmu znamenat něco jiného.

Auditable Metrics

Posledním „A“ u kvalitních metrik je dohledatelnost. U získaných dat je vždy výhodné, aby její data byla dostupná pro všechny členy týmu. Výsledek analýzy takových dat pak není možné zpochybnit – analýzu je možné kdykoli udělat znovu.

Důležité je také správně odhadnout kompromis mezi soukromím uživatelů a kvalitním měřením – v aplikaci, kde ctíme soukromí uživatelů na nejvyšší míru, není možná ani běžná analýza kohort, protože nerozeznáme ani ty uživatele, kteří již aplikaci navštívili. Opačným extrémem je pak získávání a trvalé ukládání informací o každé jednotlivé aktivitě uživatele (příkladem může být sociální síť Facebook), které sice nabízí nejvíce prostoru pro různé metriky a analýzy, existuje zde však velké riziko zneužití osobních údajů. Je tedy nutné najít střední cestu – tedy rozhodnout, která data jsou pro uživatele citlivá a nesmí se dostat ani

k členům vývojového týmu (resp. nebudou se vůbec ukládat), a která naopak mohou posloužit k vylepšení produktu, aniž by přitom znalost takových dat ve vývojovém týmu poškodila samotné uživatele.

Také je doporučeno provádět analýzy přímo na původních datech – bez prostředníků a předzpracovaných informací, minimalizuje se tak riziko chyby a zvyšuje se dohledatelnost.

4.2.5 Metriky podle Engine of Growth

Při výběru metrik je vždy třeba brát ohledy na zvolenou strategii. Konkrétně je vždy nutné sledovat alespoň jednu metriku, týkající se zvoleného způsobu získávání nových uživatelů, tedy engine of growth (viz kapitola 3.1.2 Strategie).

Při nasazení Sticky Engine of Growth, tedy opakovaného využívání stávajících zákazníků, je situace nejjednodušší – stačí sledovat celkový počet uživatelů. Pokud přichází více nových zákazníků, než odchází těch stávajících, zisky porostou.

V případě virálního šíření je problém složitější, je třeba sledovat, jak stávající uživatelé přivádějí další uživatele. Průměrný počet přivedených zákazníků by pak měl být vyšší než jedna, pak startup rychle poroste. Při průměrné hodnotě rovné jedné startup roste pouze lineárně, což pro růst startupu není dostačující, méně než jedna pak znamená postupný úpadek.

Metriky u systému, postaveného na placeném marketingu, se musí soustředit na konkrétní výši zisků. Zajímavá je pak hodnota „LTV“, lifetime value, tedy zisk z jednoho zákazníka za celou dobu, po kterou produkt využívá. Tuto hodnotu je pak třeba porovnat s hodnotou „CPA“ (cost per aquisition, cena za získání zákazníka) – a pokud je větší, startup vydělá. Do hodnoty CPA se počítá poměrná část ze všech výdajů – ne jen z reklamy, ale i vývoje či například ceny za uživatelskou podporu.

4.3 Innovation accounting

4.3.1 Cyklus učení

Postup vývoje v této fázi, za využití minimálního produktu a nových metrik zahrnuje celkem tři logické fáze:

Výchozí bod

Okamžik, kdy je vypuštěn minimální produkt mezi uživatele, můžeme označit jako výchozí bod. V prvních dnech (či týdnech nebo měsících) je nutné změřit všechny statistiky uživatelského chování, které nás zajímají (procento registrovaných atp.). Tato čísla se pro nás stanou výchozími hodnotami, cílem dalšího postupu je samozřejmě tyto hodnoty vylepšit.

Ladění

V další fázi přichází na řadu již zmiňované učení. Tým postupně vylepšuje vypuštěný minimální produkt, a po každé jednotlivé změně měří dopad této změny na sledované veličiny v chování uživatelů. Učí se tak, které změny pomáhají posouvat chování uživatelů tím směrem, který startup vyžaduje, a které změny naopak nemají vliv žádný či dokonce výsledkům uškodí.

Tým se tedy učí, jakým směrem směřovat vývoj dál.

Změna nebo pokračování

Pokud tým v této fázi vývoje zjistí, že bez ohledu na aplikované změny výsledky zůstávají neuspokojivé, je čas na změnu strategie. Tento moment se nazývá **Pivot** a může znamenat zásadní změnu v přístupu k produktu.

4.3.2 Druhy pivotů

Ze sledování chování uživatelů je možné často velmi přesně odhadnout, který typ změny strategie bude pro následující vývoj produktu výhodný.

Nemusí však zůstat u jednoho pivotu, situace se může opakovat – jedna změna strategie situaci vylepší, ale ne dostatečně, a tak po dalším cyklu učení následuje pivot další.

Můžeme rozlišit několik druhů pivotu, například:

Zoom-in Pivot

Uživatelé využívají jen jednu součást produktu, zbytek aplikace ignorují, nebo dokonce považují za přítěž. Je tedy vhodné produkt zmenšit, a zaměřit se pouze na oblíbenou část a dále rozvíjet ji. Zbytek stávajícího produktu se bohužel stává odpadem.

Zoom-out Pivot

Zřejmě nejčastější typ pivotu – stávající funkce jsou sice funkční a využívané, ale nedostatečné, je třeba produkt výrazně rozšířit. Původní produkt se tak stane jen malou částí nového produktu. Příkladem může být segment výrobců antivirových programů – na poli antivirů již není příliš místa pro inovace, proto se antiviry rozšiřují například o firewally či nástroje proti spamu.

Customer Segment Pivot

Velmi nepříjemný typ změny. Statistiky ukáží, že produkt má větší úspěch u zcela jiné cílové skupiny, než bylo původně zamýšleno. Je tedy třeba přepracovat strategii s ohledem na novou cílovou skupinu. Toto rozhodnutí může vést k mnoha malým změnám napříč produktem, které ještě více zvýší úspěšnost produktu u nové cílové skupiny. Tento typ pivotu můžeme pozorovat u herních společností, které do svých titulů stále častěji přidávají rozšířené tutorialy a systémy odměn (achievementů), aby více zaujaly nově objevenou skupinu příležitostných hráčů.

Customer Need Pivot

Zjistíme, že problém, který řešíme, zákazníky příliš nezajímá, ale existuje velmi blízký problém, který opravdu mají, a my ho můžeme vyřešit. Tomuto typu změny odpovídá například aktuální situace na trhu klientských aplikací pro instant messaging – aplikace sledují přesun uživatelů na sociální sítě, a místo nových funkcí instant messagingu se více orientují na spolupráci se sociálními sítěmi.

Platform Pivot

Při tomto typu změny se ze softwaru stává platforma či naopak. Ilustračním příkladem mohou být operační systémy mobilních telefonů – starší systémy samy obsluhovaly veškerou funkčnost telefonu, nové systémy jsou pouze platformou pro aplikace třetích stran, které tyto funkce zajišťují samy.

Business Architecture Pivot

Tento pivot znamená změnu zaměření na jiný typ zákazníků. Nejčastěji se jedná o přechod z prodeje menšího množství drahých produktů pro firmy na prodej většího

množství levnějších produktů pro malé zákazníky, či naopak. Příkladem je např. přechod firmy Bohemia Interactive Simulations z vývoje běžných počítačových her na vývoj vojenských simulátorů, jejichž kupci jsou profesionální armády.

Value Capture Pivot

Tato změna se týká Value Hypothesis. Jedná se jednoduše o změnu toho, na čem produkt vydělává. Typickým případem může být zavedení placené varianty produktu, výměnou za odstranění reklam třetích stran, na jejichž zobrazování produkt vydělával doposud.

Engine of Growth Pivot

Princip tohoto pivotu je jasný již z názvu – jde o změnu předpokladu Growth Hypothesis. Například se může jednat o přechod z virálního šíření produktu na placené formy marketingu, jak to ukázala například firma Google – po letech provozu přestala spoléhat na rozesílání pozvánek stávajících uživatelů uživatelům budoucím, a začala vytvářet televizní reklamy.

Channel Pivot

Při tomto pivotu dochází ke změně způsobu, jakým produkt prodáváme – např. změna z prodeje prostřednictvím třetí strany na přímý prodej. Do této kategorie také patří například postupný přesun prodeje počítačových her z kamenných prodejen do elektronických distribučních kanálů.

Technology pivot

Dalším typem pivotu je změna technologie – příkladem můžou být například webové aplikace, přecházející z klasického způsobu fungování na načítání pomocí technologie AJAX.

4.3.3 Obavy z pivotu

Nejčastějším důvodem selhání startupu je neschopnost či neochota provést pivot. K takovému případu dochází nejčastěji z těchto důvodů:

Nevhodné metriky

Využití takzvaných „vanity metrics“ může vést k pocitu, že se startup vyvíjí dobře a produkt je úspěšný. Pak tým jednoduše nepozná, že je pivot potřeba. Typickým případem je sledování absolutního počtu zaregistrovaných uživatelů, bez ohledu na jejich další chování. U naprosté většiny webových aplikací toto číslo neustále roste, zvlášť pokud systém nemaže dlouhodobě neaktivní účty. Tým pak získá pocit, že je produkt u uživatelů oblíbený, zatímco uživatelé se jen zaregistrují, zjistí, že jim produkt nevyhovuje, a do aplikace se již nikdy nevrátí.

Nejasná hypotéza

Tento problém nastává, pokud si tým špatně stanoví svůj cíl – ať už se jedná o součást vize, strategie, či jen cílové hodnoty sledovaných statistik pro aktuální cyklus učení. Dojde pak k situaci, kdy je hypotéza za každých podmínek splněna, nebo není možné její splnění ověřit. Tým pak nedostává potřebný impuls k pivotu.

Nenaplnění vize

Velmi častým důvodem k opomenutí pivotu je obava z vypuštění produktu, který nebude plně reprezentovat kompletní vizi. Tým pak odkládá vypuštění minimálního produktu a nezjistí, že některá ze základních hypotéz je špatná. Ve chvíli, kdy je pak produkt vypuštěn, je již na pivot pozdě – produkt neuspěje, a vzhledem k jeho velikosti je již velmi obtížné rozpoznat, která jeho část neúspěch způsobuje, a jaký typ změny by byl třeba, aby došlo k nápravě.

4.3.4 Kanban

Princip Kanban je jednou z původních součástí metodiky Lean Manufacturing, vyvinutou ve firmě Toyota. Týká se rozdělení pracovních úkolů během cyklu učení, během ladění produktu. Tento princip stanovuje, že ve vývoji může být v jednu chvíli pouze omezený počet jednotlivých změn produktu. Stanovuje také několik fází vzniku takové změny – tento počet změn se pak vztahuje na jednotlivé fáze, tedy v jedné fázi vývoje nemůže být v jednu chvíli více než stanovený počet změn – pracovních úkolů.

Nejjednodušší verzi principu Kanban zobrazuje následující obrázek:

TODO	PROBÍHÁ max 3	NASAZENO max 3	OVĚŘENO
E F G H	A B C	D J I	K

Úkoly-změny jsou rozděleny do čtyř základních skupin – první sloupec je zásobníkem změn, které jsou v plánu, druhý sloupec obsahuje ty změny, na kterých tým právě pracuje. Ve třetím sloupci jsou ty změny, které již byly nasazeny mezi uživatele, a v posledním sloupci jsou změny, které byly ověřeny – tedy tým ví, jaký byl jejich dopad na chování uživatelů.

Ve chvíli, kdy bude úkol A dokončen, nesmí být ihned nasazen, protože ve třetí fázi již tři změny jsou. Stejně tak nesmí být zahájena práce na další změně z prvního sloupce. Tento princip donutí členy týmu, kteří nesmí zahájit práci na dalším úkolu, k neprogramátorským aktivitám, konkrétně ověření účinnosti tří změn ve třetím sloupci. Tým tak může nasadit A/B testování, běžnou analýzu kohort, či osobně kontaktovat uživatele, a ověřit tak dopady nasazení této změny. Pak teprve může tuto nasazenou změnu přesunout do posledního sloupce, a tím uvolnit místo úkolům dalším, na nichž teď teprve může začít tým pracovat.

Tento přístup tedy zajistí, že bude analýza provedena na všech dokončených úkolech a tým tedy vždy dokončí cyklus učení – vždy zjistí, jaký následek změna měla, a jestli bylo výhodné ji vytvořit či ne.

4.3.5 Pět „Proč“

Hlavním principem metodiky Lean Development je učení – tým se učí, které změny produktu prospívají a které ne, učí se zlepšovat svůj produkt. Učit se tým může i z vlastních chyb a tím může zvyšovat svou produktivitu. Pomáhá při tom princip Pěti Proč, který taktéž pochází z firmy Toyota jako součást metodiky Lean Manufacturing.

Princip této metody je jednoduchý – vždy, když dojde k chybě (v produktu, ve vývoji, kdekoli), zeptáme se, proč k tomu došlo. Když najdeme příčinu chyby, zeptáme se na příčinu této příčiny atd., celkem pětkrát. Tento na první pohled podivný způsob dokáže často odhalit nečekané problémy, o kterých by se tým jinak nedozvěděl. Užitečnost můžeme ukázat na příkladu:

- Při nasazení nové verze aplikace přestala fungovat jedna funkce.
- **Proč?**
- V databázi byly neočekávané hodnoty NULL.
- **Proč?**
- Do databáze byl vložen nový sloupec bez zadané výchozí hodnoty.
- **Proč?**
- Nástroj na správu databáze neumožňuje při přidávání sloupce výchozí hodnoty nastavit.
- **Proč?**
- Používáme zastaralou verzi nástroje na správu databáze.
- **Proč?**
- Odpovědný člen týmu nástroj zapomněl aktualizovat.

Při běžném postupu, hledajícím jen příčinu prvotního problému, by tým pouze opravil databázi – vložil by správné hodnoty. Chyba by se ale časem opakovala, při vložení nových řádků. Při hlubší kontrole by možná byly opraveny i výchozí hodnoty. I tak by ale brzy nastal problém znovu – u nových sloupců. Systém pěti Proč tedy zařídí, že se navíc aktualizuje nástroj na správu databáze, případně se zavedou pravidla pro jeho další aktualizaci.

Tento přístup tak výrazně zmenšuje riziko, že se stejná či podobná chyba bude opakovat.

5 Vývoj Infinity I – Vize a strategie

Jak již bylo zmíněno v úvodu, projekt Infinity je webová aplikace, vyvíjená týmem studentů MFF UK v rámci předmětu Softwarový projekt.

Tento projekt byl vyvíjen bez znalosti metodiky Lean Development, přesto se však tým v některých fázích projektu této metodice nevědomky přiblížil.

Další kapitoly budou popisovat vývoj tohoto projektu v chronologickém pořadí.

Tato kapitola se týká zvolení vize a strategie projektu – v této fázi se tým nevědomky takřka stoprocentně shodnul s metodikou Lean Development – stanovil vizi a obě pro strategii klíčové hypotézy, hypotézy hodnoty a růstu, a částečně obhájil jejich pravdivost.

5.1 Vize Infinity

Vize Infinity byla ve své podstatě jednoduchá. Výsledkem projektu Infinity měla být webová hra, situovaná do sci-fi prostředí, která spojí dvě různé cílové skupiny a bude obsahovat některé unikátní prvky.

5.1.1 Cílové skupiny uživatelů

Projekt Infinity si vzal za cíl přilákat dvě velmi nesourodé cílové skupiny uživatelů – hráčů.

První cílovou skupinou jsou **příležitostní hráči** (v angličtině Casual Players). Do této skupiny spadají lidé, kteří příliš často hry nehrají, a ještě méně často je kupují. Pokud je nějaká hra zaujme, je často velmi jednoduchá a obsahuje snadno pochopitelné repetitivní principy. Do této skupiny spadají například ženy a děti do 12 let.

Druhou, naprosto odlišnou cílovou skupinou, jsou **zkušení hráči** webových strategických her, kteří vyžadují propracovanější a komplikovanější herní prvky, stejně jako vzájemné zápolení.

5.1.2 Unikátní herní prvky

Hlavní výhodou Infinity ve srovnání s konkurenčními hrami měl být systém Poddanství, který žádná jiná hra nenabízí – v tomto systému se poražený hráč stává poddaným vítěze. Hlavní výhodou tohoto systému je absence destrukce – vítěz získá poddaného a je spokojen, aniž by poddaný musel přijít o svou pracně budovanou planetu.

Druhým, méně originálním prvkem, který měl přilákat hlavně zkušenější hráče, mělo být sestavování kosmických lodí z komponent, každá loď tak měla být unikátní.

5.2 Strategie Infinity

5.2.1 Value hypothesis

Nejprve je nutné zjistit, zda existují uživatelé, pro které projekt Infinity bude určen, a zda budou ochotni produkt používat. Je tedy nutné stanovit základní předpoklady a alespoň teoreticky obhájit jejich pravdivost. Stejně tak je nutné rozhodnout, jakým způsobem bude hra vydělávat.

Webová aplikace a Facebook

Tým Infinity rozhodl, že hra bude webovou aplikací, přístupnou všem hráčům pouze prostřednictvím jejich webového prohlížeče. Důvod je jednoduchý – přístup k internetu mají v dnešní době dvě třetiny populace ČR (3) a na celém světě cca třetina lidí (4), a pro většinu hráčů je jednodušší spustit hru přímo v prohlížeči, než instalovat do počítače samostatný software. Jednodušší je pak také správa aplikace – všichni uživatelé mají zaručeně aktuální verzi, nad kterou má vývojářský tým plnou kontrolu.

Projekt Infinity vznikl (obdobně jako konkurenční hry) jako aplikace pro platformu serveru Facebook.com – ten zajišťuje autentizaci a autorizaci uživatele (a software tedy nemusí sám řešit registraci uživatelů), prostřednictvím API poskytuje další užitečná data o uživateli (přátelé uživatele) a usnadňuje propagaci produktu.

První cílová skupina

V době, kdy vznikala Vize Infinity, měly hry, zaměřené primárně na tento typ hráčů, velký úspěch. Využívaly většinou platformy serveru Facebook.com, na kterém měli tito hráči ke hrám velmi snadný přístup. Nejúspěšnější hrou té doby byl FarmVille, který měsíčně hrálo více než sto milionů uživatelů po celém světě, a desítky dalších her překročily počet jednoho milionu aktivních hráčů.

Tyto hry měly většinou společné základní prvky – opakoval se v nich jednoduchý základní cyklus (např. ve FarmVille: nakup, zasaď, počkej, skliď a prodej). Dalším společným jmenovatelem byla absence konce hry či jiných běžných herních cílů. Hráč jednoduše

vydělával více a více herních peněz, za ně kupoval dražší a dražší herní prvky, a vše se stupňovalo do nekonečna. Jedinou motivací tak byla samotná hra, která u cílové skupiny projevovala značnou návykovost, a také soupeření s přáteli – platforma aplikace Facebook umožňovala snadné porovnávání s přáteli, a snaha překonat své přátele tak byla hnacím motorem pro velké množství hráčů.

Pro skupinu příležitostných hráčů je určena první fáze hry Infinity, ve které hráči budují na své planetě město. Fáze využívá oblíbené a vyzkoušené principy z konkurenčních her, například jednoduché cykly (zakoupit budovu, počkat na dostavění, vydělat na jejím provozu), a podobně jako hry od konkurence je čistě mírumilovná a sází pouze na návykovost a porovnávání mezi přáteli.

Tato fáze obsahuje budování města na planetě hráče, spolu s udržováním spokojenosti jeho obyvatel – sci-fi prvky v této fázi jsou minimální.

Příležitostných hráčů se na sociální síti Facebook vyskytují stovky milionů, jak ostatně napovídají počty uživatelů konkurenčních her pro tuto skupinu. Žebříček nejpopulárnějších her obsahuje značné množství klonů těch nejlepších her, či her, které jinak recyklují zaběhnuté postupy. Mohli jsme tedy usuzovat, že hra, která bude tyto principy zahrnovat, dokáže vybudovat dostatečnou základnu uživatelů.

Druhá cílová skupina

Na českém trhu existovalo několik webových her, které měly velký úspěch v letech předcházejících vzniku Infinity. Úspěšná byla například německá hra Travian, ale v českém prostředí získaly nejvíce pozornosti původní české webové hry (Webgame, RedDragon, Darkelf a další). Počty jejich hráčů se pohybovaly v řádech desítek tisíc, což je na produkty, zaměřené pouze na český trh, relativně velký úspěch.

V době vzniku Vize Infinity už uplynul dlouhý čas od spuštění posledních populárních webových her, a podle rozhovorů s mnoha hráči z této skupiny byl na trhu velký prostor pro novou hru.

Hra, mířící na tuto skupinu, nemohla uspět s mírumilovnými repetitivními principy, které zaujaly první skupinu. Všechny konkurenční hry z této skupiny obsahovaly nějakou formu přímého boje mezi hráči, a většinou také nebyly nekonečné. Byly rozděleny do tzv.

věků, které trvaly od několika dnů až po několik měsíců, a každý věk měl svého nezpochybnitelného vítěze, který se svou výhrou zapsal do historie své hry. Toto vítězství bylo samozřejmě hlavní motivací pro tuto skupinu hráčů.

Pro zkušené stratégy měla vzniknout druhá fáze hry Infinity. V ní hráči, kteří už dříve v první fázi vystavěli město na své planetě, tuto planetu opustili a v okolním vesmíru bojovali o nadvládu s ostatními hráči.

Druhá fáze měla umožňovat sestavování vlastních kosmických lodí (ze zvolených komponent), cestování mezi planetami, obchodování mezi hráči a v neposlední řadě boje.

Zkušenější hráči na rozdíl od těch příležitostných na sociální síti Facebook často své uplatnění postrádali a z osobního průzkumu vyšlo najevo, že novou hru uvítají. Tito hráči navíc spadají mezi tzv. „early adopters“, tedy skupinu uživatelů, kteří rádi zkouší nové možnosti, nové produkty a nové aplikace.

Prolnutí skupin

Zbývalo jen posoudit, jestli se jednotlivé fáze hry, zaměřené na různé skupiny hráčů, nebudou vzájemně rušit.

První fáze měla být přizpůsobena tak, aby pro zkušené hráče bylo relativně snadné překonat ji v rychlém čase, a nadále se věnovat rozsáhlé druhé fázi, určené pro ně.

V případě druhé fáze vstoupil do hry princip poddanství. Ten měl být nastaven tak, aby poddaný o nic nepřicházel, a nadřízený ze svého postavení profitoval. Příležitostní hráči, kteří nemuseli do druhé fáze vůbec postoupit, tak mohli být součástí boje ve druhé fázi – jejich planety mohl získat některý z útočících hráčů jako poddané, aniž by o tom sami poddaní hráči museli vědět (dokud nepostoupili sami do druhé fáze).

Výdělek

Bylo rozhodnuto, že tým využije principu mikrotransakcí. Při využití této metody je samotné hraní zdarma, hráč však může za reálné peníze dokoupit do hry jisté herní výhody – herní prvky, které se běžným způsobem nedají vůbec sehnat, nebo jen velmi obtížně. Tento způsob úspěšně funguje u většiny konkurenčních her v prostředí sociální sítě Facebook.

5.2.2 Growth hypothesis

Projekt Infinity vsadil na variantu „Viral engine of growth“, tedy **virální šíření**. V prostředí sociální sítě Facebook je to vyzkoušený způsob, na který už jsou zvyklí i sami uživatelé – běžně zvou své přátele do hry, kterou sami hrají, a sami přijímají pozvánky svých přátel. Jako doplněk pak slouží propagace na samotné sociální síti Facebook (výskyt hry v katalogu her) a samozřejmě ústní šíření, přímé doporučování od hráčů.

6 Vývoj Infinity II – Tým a technologie

Sestavení týmu a výběr technologií jsou nejdůležitějšími rozhodnutími pro celý vývoj produktu.

Změna technologie (obzvláště v pozdější fázi vývoje) má za následek velké množství „odpadu“, kterému se jako vývojáři chceme vyhnout.

Následující podkapitoly popíší dostupné technologie pro vývoj webové aplikace použitelné v současnosti (rok 2012) a blízké budoucnosti, s větším důrazem na technologie, které zvolil tým Infinity.

6.1 Tým

V oblasti sestavení vývojářského týmu nepřináší přístup Lean Development žádná nová pravidla – jen dále zdůrazňuje některá již zaběhnutá.

Typickým příkladem (pro webovou aplikaci) je například specializace vývojářů na front-end, resp. back-end. Toto rozdělení urychlí vypouštění jednotlivých nových funkcí v produktu, a tedy bude dříve dostupná i odezva a měření dopadů těchto změn.

Tým Infinity vsadil na toto rozdělení – dva programátoři se věnovali back-endu, dva front-endu a grafickému rozhraní. Toto rozložení se osvědčilo - každou novou funkčnost měl na svědomí vždy jeden programátor front-endu a jeden programátor back-endu, druzí dva mohli následně jejich práci kontrolovat. Na týdenních poradách se pak celý tým seznámil zběžně s novým kódem, aby byl v případě nutnosti každý schopen aspoň základní orientace a jednoduchých úprav v kompletním kódu, back-endu i front-endu dohromady.

Poslední částí týmu Infinity byl externí grafik.

6.2 Nástroje pro týmovou spolupráci

Při spolupráci více vývojářů nelze spoléhat jen na konvenční metody komunikace, jako jsou e-maily, instant messaging či videohovory. Ty se dají využít na základní domluvu při každodenních činnostech či menších poradách, ale vždy je nutné rozhodnutí zdokumentovat, nejlépe tak, aby byla vždy dostupná a editovatelná pro všechny členy týmu.

Taková rozhodnutí mohou zahrnovat plány do budoucna, designová rozhodnutí a jejich zdůvodnění, ale také každodenní rozdělení úkolů v týmu.

Možností, jak takovou vývojovou dokumentaci udržovat je mnoho a není snadné vždy zvolit nejefektivnější z nich. V tomto případě opět přichází ke slovu Lean Development, který o „vhodném řešení“ mluví sice v kontextu rozhodnutí o designu produktu, ale toto tvrzení se dá aplikovat i na výběr technologií a v tomto případě nástrojů pro týmovou spolupráci. Toto tvrzení říká, že je velmi důležité poznat, kdy stávající přístup (v tomto případě nástroj) nevyhovuje – v takové chvíli je důležité (i když mnohdy obtížné) nástroj opustit a zvolit takový, který lépe odpovídá současným podmínkám.

Tým Infinity využil postupně velkou škálu těchto nástrojů.

V první fázi, kdy teprve vznikala vize projektu a základní myšlenky strategie, se osvědčily kancelářské online nástroje, jako jsou **Dokumenty Google** (Google Docs). V této fázi spočívala hlavní výhoda Docs v možnosti upravovat jeden dokument všemi členy týmu najednou, zatímco ostatní viděli jejich úpravy v reálném čase. Tvorba takového dokumentu navíc probíhá přirozeně, bez nutnosti učit se zacházet s novými mechanismy. Tým tak mohl rychle a efektivně navrhovat nové možnosti, na místě o nich diskutovat a následně ponechat v dokumentu jen výsledná rozhodnutí a jejich zdůvodnění.

Později, se začátkem programování, se tento dokument už nevyvíjel živelně a nevyžadoval současný přístup více lidí najednou, na druhé straně stoupala poptávka po větší strukturovanosti a přehlednosti. Rozhodnutí o designu se tedy přesunula z online dokumentu do dalšího nástroje – **vlastní WIKI**. Nástrojů pro tvorbu wiki je velké množství (MediaWiki, DokuWiki…) a všechny nabízely právě to, co tehdy tým potřeboval – strukturované úložiště „plánů“ vznikajícího produktu, nápadů na nové vlastnosti a zdůvodnění těchto návrhů. Zvolena byla DokuWiki, ale hlavním důvodem byly pouze předchozí zkušenosti členů týmu s DokuWiki.

Zároveň vznikla potřeba nástroje pro rozdělení úkolů. První rozdělení proběhlo na nové wiki (což se záhy ukázalo jako zoufale neefektivní) a velmi brzy následoval přesun na plnohodnotný **Issue Tracker,** tedy nástroj zaměřený přímo na správu úkolů (který tým získal zdarma a bezpracně v rámci webového repozitáře verzovacího systému).

Později se ukázalo, že hlavní výhody tohoto nástroje (perzistence úkolů i po jejich dokončení, možnost zadávání úkolů/problémů cizími lidmi, více stavů úkolu atd.) tým nevyužije, a poněkud komplikovaný nástroj se tak stává pouhým odškrtávacím seznamem úkolů. Práce v něm je navíc do jisté míry zatížena obtížností práce s tak robustním nástrojem. Víceméně neúmyslně a přirozeně se tak seznam úkolů přesunul zpět na Google Docs, kde všichni členové týmu viděli úkoly všech kolegů, mohli se k nim vyjadřovat či je například přesouvat mezi jednotlivými vývojáři. Tento přístup nemusí vyhovovat každému týmu, v projektu Infinity se ale osvědčil a je dobré vědět, že i tak jednoduše může probíhat správa úkolů v týmu.

6.3 Technologie pro Back-end

6.3.1 Webový server

Základem webové aplikace je samozřejmě webový server. V současnosti existuje velké množství variant, pro unixové/linuxové servery i pro servery s operačním systémem Windows.

Nejznámější variantou je program **Apache HTTP Server**. Toto open source řešení je momentálně nejvíce rozšířeným web-serverem (přes 64% serverů na internetu (5)) a je dostupné pro většinu existujících operačních systémů.

Hlavním soupeřem Apache na poli webových serverů je **IIS** od Microsoftu, uzavřené řešení pro servery s OS Windows, vhodné hlavně pro využití ve spolupráci s ostatními technologiemi firmy Microsoft, jako jsou .NET, MS SQL Server a další.

Třetí nejrozšířenější variantou, kterou zvolil i tým Infinity, je server **Nginx**. Stejně jako Apache, i Nginx se řadí do kategorie open source a existuje ve variantách pro více operačních systémů, včetně Windows a Linuxu. Podle nezávislých testů (například (6)) je Nginx rychlejší a výrazně méně náročný na hardware, než Apache. Navíc od začátku vývoje počítá s možností rozdělení na více fyzických serverů a je tak připraven i na velmi náročné webové aplikace.

Jedinou nevýhodou serveru Nginx je jeho relativní mládí – software vznikl v roce 2004 v Rusku a větší úspěch v anglicky mluvícím světě zaznamenal teprve v posledních letech, proto k němu není k dispozici tak velké množství návodů a tipů, jak tomu je u konkurenčního

Apache, který vznikl už v roce 1995 v USA. Tento handicap však Nginx v poslední době pozvolna překonává a stává se tak plnohodnotnou alternativou pro Apache.

Spolu s webovým serverem Nginx byla využita navíc cache Memcached – namísto session jako úložiště aktuálních dat.

6.3.2 Programovací jazyk pro back-end

Mezi programovacími (skriptovacími) jazyky pro back-end je výběr ještě větší. Nejdostupnější technologií (kterou zvolil i tým Infinity) je jazyk **PHP**, který je podle statistik (například (7)) zdaleka nejrozšířenější alternativou. S tím je spojeno také velké množství návodů, tipů, ale i knihoven a doplňků, stejně jako kvalitní manuál a široká komunita. Velkým kladem při výběru byl také fakt, že všichni členové týmu Infinity již PHP znali a byli schopni v něm ihned začít pracovat.

Alternativou je znovu technologie Microsoftu, **ASP.NET**, jejíž hlavní výhodou je opět dobré propojení s ostatními technologiemi této firmy, nevýhodou pak nedostatek webhostingů zdarma (a tedy menší možnosti testování) a obecně cena kompletního serverového řešení Microsoftu (zvlášť ve srovnání s alternativami, které jsou kompletně zdarma).

Třetí místo ve výše uvedeném žebříčku nejpoužívanějších serverových programovacích jazyků obsadila **Java**. Tuto technologii tým Infinity taktéž zvažoval – existují serverová řešení zdarma (stejně jako u PHP) a podpora v komunitě vývojářů či existence manuálů jsou jasné výhody. Nevýhodou je ale hlavně menší výkon ve srovnání s většinou alternativ.

Další variantou, která bude zajímavá hlavně v blízké budoucnosti, je využití Javascriptu, konkrétně serverového řešení **Node.js**. Tato mladá alternativa však zatím není dostatečně vyzkoušená, a přestože se rychle rozvíjí, pro velké projekty ji zatím není možné doporučit.

Méně častými alternativami jsou pak jazyky **Perl**, **Python** a **Ruby**, jejich rozšíření se ale dlouhodobě pohybuje okolo úrovně jednoho procenta a níže a tým Infinity je při výběru technologií nebral v úvahu, zvláště proto, že nikdo z týmu s jejich využitím neměl zkušenosti.

6.3.3 Databáze

Webová aplikace ve většině případů vyžaduje využití databázového serveru pro ukládání persistentních dat uživatelů a aplikace samotné. V této kategorii v poslední době probíhají velké změny, starší SQL technologie částečně ustupují některým No-SQL variantám.

Mezi SQL servery můžeme rozlišit placené technologie a technologie zdarma. Do první kategorie spadají hlavně databázová řešení společností **Microsoft** a **Oracle**, ale i další. Tyto varianty jsou dostatečně vyzkoušené a zdokumentované, jejich hlavní nevýhodou je tak jejich cena a možná až zbytečná komplikovanost pro potřeby webové aplikace.

MySQL je nejznámější SQL technologií mezi bezplatnými řešeními, jeho problém je pak naprosto opačný: přílišná jednoduchost a například malá připravenost na velkou zátěž – pro rozdělení na více serverů je potřeba využít další technologie, například MySQL Cluster.

Tým Infinity nakonec vybíral mezi několika No-SQL variantami. První (nakonec zamítnutou) byla **Cassandra**. Tuto technologii vyvinul v roce 2008 Facebook jako open source, ale sám ji o dva roky později opustil a přenechal organizaci Apache. Cassandra sází na maximální jednoduchost a systém klíč-hodnota, je vytvořena pro práci na více serverech, kde každý je zastupitelný – není tak možné databázi znepřístupnit poškozením jednoho konkrétního serveru.

Nakonec tým Infinity zvolil databázi **MongoDB**. Její víceserverové řešení sice není tak robustní jako v případě Cassandry, zato je snáze nastavitelné. Stejně tak z pohledu programátora je Mongo velmi přívětivé. Databáze sestává z kolekcí (analogie SQL tabulek), obsahujících jednotlivé dokumenty (analogie SQL řádků). Na rozdíl od SQL ale dokumenty/řádky nemají předem zadané pevné schéma – mohou obsahovat libovolně strukturované asociativní pole, včetně vnořených struktur, dokonce různé dokumenty v jedné kolekci mohou mít naprosto odlišné struktury, společným prvkem je jen ID každého dokumentu. Mezi zajímavé vlastnosti Monga pak patří například prostorové indexy – dokumenty mají určenu pozici v euklidovském prostoru, a pomocí indexu je pak možné vybírat dokumenty, které se nachází v určité vzdálenosti od zadaných souřadnic atp. Mongo existuje ve variantách pro Windows i Linux a spolupracuje s PHP – PHP obsahuje nativní ORM API pro práci se strukturami Monga.

6.4 Technologie pro Front-end

6.4.1 HTML

Na první pohled se může zdát, že v této kategorii je třeba zvolit mezi dostupnými variantami HTML – na výběr je staré, ale stále nejrozšířenější HTML 4, dále novější varianta, postavená na XML serializaci – XHTML, a v neposlední řadě nová, teprve nastupující specifikace HTML5.

Ve skutečnosti však je tento výběr irelevantní – žádný z dnešních prohlížečů mezi těmito variantami nerozlišuje a ve všech případech používá stejný parser a stejné vykreslovací jádro. Dnešní prohlížeče nečtou definici DOCTYPE, pouze detekují jeho přítomnost (v jeho nepřítomnosti zapínají starý vykreslovací mód, známý jako Quirks mode). Ideální je proto použití nového a jednoduchého DOCTYPE html5, který vypadá takto:

```
<!DOCTYPE html>
```

Výběr je tedy nutné posunout na jinou úroveň – na úroveň jednotlivých částí jazyka HTML. Aktuální prohlížeče postupně přidávají podporu dalších a dalších prvků technologie HTML5, proto je nutné sledovat konkrétně podporu jednotlivých funkčností. Existuje naštěstí hned několik webových stránek, které aktuální podporu HTML5 v prohlížečích sledují – mezi nejznámější patří například web caniuse.com.

Mezi nejviditelnější novinky v HTML5 patří nové sémantické elementy, například footer, header a další. Jejich využití je v dnešní době takřka bezproblémové – všechny prohlížeče mimo Internet Explorer je buď znají, nebo se k nim chovají jako ke všem neznámým elementům – tedy stejně jako k univerzálnímu elementu DIV. Jelikož nesou pouze sémantickou informaci, tyto dva případy se vůbec neliší. Internet Explorer tyto elementy ignoruje, ale existuje hned několik javascriptových skriptů, které jejich podporu do libovolného prohlížeče Internet Explorer dodávají. Projekt Infinity tedy těchto elementů využil.

Horší situace je u formulářových prvků, jejich specifikace byla několikrát z HTML5 vyňata (a osamostatněna jako WebForms 2.0) a opět přidána. Proto je podporuje jen malé množství prohlížečů (například Opera). Použití této technologie v současnosti není možné doporučit, ale v budoucnosti se bude jednat o velmi užitečnou technologii – specifikace například obsahuje automatickou validaci hodnot formulářových prvků (například kontrola, zda řetězec v poli „e-mail“ opravdu odpovídá e-mailové adrese), což je vlastnost, kterou

v dnešní době musí řešit programátor, a hned na dvou úrovních (před odesláním pomocí javascriptu, po odeslání v serverovém skriptovacím jazyce). Dále specifikace obsahuje např. speciální formulářové prvky pro výběr data, čísla z daného rozsahu a mnoho dalších novinek.

Z dalších částí HTML5 stojí za zmínku ještě Canvas. Tento nový element slouží jako plátno, na které je možno kreslit pomocí javascriptu. Jako technologická dema byly pomocí canvasu vytvořeny například různé 3D hry (např. přesná kopie staré hry Wolfenstein 3D). Podpora Canvasu v současných prohlížečích však zatím pokulhává. Internet Explorer do verze 8 Canvas nepodporuje vůbec, ostatní prohlížeče se liší hlavně podporou pokročilejších funkcí (8). Tým Infinity provedl několik experimentů, týkajících se použitelnosti alespoň základních funkcí Canvasu, ukázalo se ale, že prohlížeče nevykreslují operace s Canvasem příliš optimálně – ve všech testovacích případech se nakonec ukázalo vhodnější „kreslit“ pomocí vytváření samostatných HTML elementů než modernější kreslení do Canvasu.

Specifikace HTML5 obsahuje ještě velké množství dalších prvků, ty však buď nemají dostatečnou podporu v prohlížečích, nebo je tým Infinity nepotřeboval k práci na projektu, a proto se o nich tato práce nebude zmiňovat. Patří mezi ně například elementy Video a Audio, podpora lokálního úložiště nebo Cross-document messaging.

6.4.2 CSS

V kategorii CSS je situace velmi podobná situaci ohledně HTML. Různé nové verze technologií (CSS 2.1, CSS 3.0) jsou v různých stádiích implementace jednotlivými prohlížeči, a nemá smysl je vnímat jako celek. Je opět třeba zajímat se o aktuální podporu jednotlivých funkčností, opět pomáhají weby jako caniuse.com.

Z nových vlastností jsou některé již v dnešní době použitelné ve všech prohlížečích – kulaté rohy, stíny textu či stíny elementů, a další. Někdy je však potřeba použít více variant CSS pravidla – oficiální variantu a navíc ještě prefixované varianty pro ty prohlížeče, které sice již vlastnost implementovaly, ale ještě se 100% neshodují se standardizovanou verzí a proto uchovávají svou implementaci pod jiným jménem (typickým příkladem je vlastnost border-radius, -moz-border-radius pro starší prohlížeče Firefox či –webkit-border-radius pro starší verze Chrome či Safari).

Tým Infinity využil ve hře nové CSS vlastnosti jen minimálně a jen v případech, že měly stoprocentní podporu na cílových prohlížečích (IE7 a novější, všechny moderní

prohlížeče). Na informačním webu, který byl součástí projektu, ale ne samotné hry Infinity, byly využity i některé pokročilejší možnosti (např. CSS animace a transformace), ale vždy tak, aby jejich absence ve starších prohlížečích neznehodnotila samotnou prezentaci.

CSS Preprocesory

Další zajímavou technologií na poli CSS je CSS preprocesing. Jedná se jakékoli zpracování CSS souborů přímo na serveru, před samotným odesláním do prohlížeče. Nejjednodušším případem (který využil i tým Infinity) je spojení všech jednotlivých CSS souborů do jednoho. Důvod je zřejmý – při programování je výhodné mít CSS rozdělené do více souborů kvůli přehlednosti, při nasazení je pak lepší jeden velký soubor, kvůli menšímu počtu požadavků na server. Projekt Infinity využil vlastní jednoduchý preprocesor, který kromě spojování souborů ještě prováděl základní čištění – odstraňoval komentáře a prázdné řádky, tento preprocesing se prováděl vždy při deploymentu, tedy při každém nahrání nové verze zdrojových kódů na produkční server. Zajímavým poznatkem při testování preprocesoru byl fakt, že prohlížeče Internet Explorer (včetně nejnovějších verzí) na rozdíl od všech ostatních prohlížečů podporují pouze 32 souborů CSS. Všechny ostatní soubory IE jednoduše ignoruje – i proto byl preprocesing nezbytný.

Existuje několik známých CSS preprocesorů, které navíc plní i další úkony. Mezi jejich hlavní výhody patří CSS-proměnné (například pro uložení barvy, která je následně použita na mnoha místech v CSS souboru), základní aritmetika v CSS hodnotách, nebo tzv. nesting neboli vnořování CSS definic. Následující dva zápisy jsou ekvivalentní:

Běžné CSS

```
#red{
    color: red;
}
#red h3{
    margin: 0;
}
```

Nesting v CSS preprocesoru

```
#red{
    color: red;
    h3{
        margin: 0;
    }
}
```

Mezi další výhody patří tzv. mixiny, které umožňují určitou část CSS definice opakovaně využít na více místech, bez zbytečného opakování. Tyto mixiny navíc mohou být parametrizované s definovanou výchozí hodnotou. Následující dva zápisy jsou ekvivalentní:

Běžné CSS

```
#round1{
    border-radius: 1px;
    -moz-border-radius: 1px;
}
#round2{
    border-radius: 2px;
    -moz-border-radius: 2px;
}
```

Mixiny v CSS preprocesoru

```
.rounded-corners (@radius: 1px) {
  border-radius: @radius;
  -moz-border-radius: @radius;
}
#round1{
    .rounded-corners;
}
#round2{
     .rounded-corners(2px);
}
```

Mezi nejznámější CSS preprocesory patří **LESS**, **SASS** a **Stylus**. Všechny tři technologie existují jen několik let a stále se vyvíjí. Většího rozšíření se dočkaly teprve v poslední době, a proto je tým Infinity nevyužil. Do budoucna se ale jedná o slibná řešení, která značně usnadní práci.

6.4.3 Javascript

Programování front-endu je zřejmě jediná technologie, kde neexistuje konkurence a Javascript je jasnou volbou. Tým Infinity Javascript použil pro programování celého front-endu – od kompletní prezentační vrstvy aplikace, přes část herní logiky a obsluhu událostí až po úložiště dočasných dat.

Frameworky a knihovny

V poslední době zažívají velký boom různé javascriptové knihovny a frameworky, které programátorovi usnadňují práci – některé usnadňují konkrétní operaci, jiné zvládají velké spektrum metod, od základních operací s DOM, až po pokročilé funkce.

Na poli univerzálních frameworků panuje velká konkurence, mezi nejznámější frameworky patří **jQuery**, **Prototype** nebo **MooTools**. Všechny obsahují základní metody pro usnadnění práce s elementy v HTML dokumentu, a podobné sady doplňkových funkcí. Nejrozšířenější knihovnou z této kategorie je jQuery (9), tuto knihovnu zvolil i tým Infinity.

Hlavním úkolem jQuery je usnadnění práce s DOM. Základní objekt/funkce (v Javascriptu platí objekt=funkce) tohoto frameworku má jednoznakový název „$“ a je jediným globálním přístupovým bodem frameworku, vše ostatní je řešeno pomocí metod tohoto objektu.

Příklad:

```
$(”#red > .link”)
.css({color: ”red”, background: ”black”})
.append(”<a href=’/’>link</a>”);
```

Tento kód vybere elementy s třídou “link”, které jsou přímými potomky element s ID “red”, nastaví jim barvu a pozadí a vloží za každý z nich do DOM nový element. Zajímavostí jQuery je fakt, že všechny metody, které nepotřebují vracet hodnotu, vrací jako návratovou hodnotu samotný objekt, na kterém byly zavolány. Je tedy možné (jak je vidět na příkladu) volání metod řetězit.

Další zajímavou knihovnou, kterou tým Infinity využil, je **head.js** – tato knihovna slouží k asynchronnímu načítání dalších souborů z webu. Projekt Infinity tuto knihovnu využil k načítání všech ostatních souborů javascriptu – klasickým způsobem (přes element SCRIPT v HTML) byla vložena pouze tato knihovna a její volání, všechny ostatní soubory (veškerou herní logiku, obsluhu událostí atd.) následně tato knihovna načetla a vložila na pozadí, přičemž už prohlížeč mohl zobrazovat další části aplikace, aniž by musel čekat na načtení všech souborů, které v tu chvíli ještě nepotřeboval.

Komunikace s back-endem

Pro komunikaci s back-endem byla v projektu Infinity využita technologie AJAX, která dokáže posílat http požadavky na pozadí aplikace, bez nového načítání celé webové stránky. Tento přístup se osvědčil – ve front-endu je možné uchovávat relativně velké množství dat a s back-endem pak komunikace probíhá jen v nejnutnějších případech, typicky při zásazích do databáze. K přenášení dat byl využit formát JSON – jedná se o textovou serializaci asociativního pole, funkce na převod mezi tímto polem a textem jsou dostupné jak v PHP, tak ve frameworku jQuery, takže bylo snadné přenést strukturované pole hodnot z PHP do Javascriptu.

6.5 IDE

Kvalitní IDE, neboli Integrated development environment (integrované vývojové prostředí) je dnes pro vývojáře nezbytností. Na rozdíl od vývoje v jednoduchém textovém editoru nabízí moderní IDE mnoho funkcí, které usnadňují vývoj softwaru. Základem je zvýrazňování syntaxe kódu, případně jeho napovídání a doplňování. Velkou výhodou je i zobrazování dokumentace – IDE čte speciální dokumentační komentáře (JavaDoc, PHPDoc apod.) a následně například u volání funkcí dokáže napovídat parametry či zobrazit celou příslušnou část dokumentace. Dalšími výhodami můžou být automatizované nástroje na hledání chyb či integrace s verzovacím systémem a mnoho dalších.

Na trhu je jen málo IDE, která jsou zaměřena na vývoj webových aplikací, a tedy zvládají celou potřebnou škálu jazyků a technologií – PHP, Javascript, CSS, HTML a případně další. Nejsilnější v tomto směru jsou univerzální nástroje **Eclipse** a **NetBeans**. Dále jsou k dispozici profesionální nástroje, jako je například **PhpStorm**, ten ale na rozdíl od Eclipse a NetBeans neposkytuje verzi zdarma a proto byl z výběru vyloučen. Další placenou variantou je pak **Zend Studio**. Tým Infinity nakonec zvolil NetBeans – jsou dlouhodobě stabilnější než Eclipse a podporují všechny zvolené technologie, od HTML až po verzovací systém (viz následující kapitola 6.6).

6.6 Verzovací systém

Poslední technologií, bez které se rozsáhlý projekt nemůže obejít, je systém pro správu verzí zdrojového kódu. Schopnost koordinovat práci více programátorů, sdílet zdrojový kód mezi nimi a mít kdykoli možnost vrátit se ke starší verzi kódu, je v dnešní době naprosto nezbytná.

V úvahu připadalo hned několik verzovacích systémů.

CVS je nejstarším z doposud používaných systémů, a již nesplňuje požadavky na moderní verzovací systém, proto byl ihned zamítnut. Jeho mladší nástupce, systém Subversion (**SVN**) je již lépe připravený na současné projekty a vývoj více programátorů najednou, ale i tak pomalu ustupuje novějším, distribuovaným systémům.

Výběr se tedy zmenšil na rozšířené distribuované verzovací systémy současnosti – **Git**, **Mercurial**, a **Bazaar**. Všechny pracují na podobném systému – každý programátor má na

svém stroji vlastní kompletní kopii historie projektu a všechny změny kódu. Tuto lokální kopii pak synchronizuje s hlavním úložištěm, do kterého své změny tímto způsobem vkládají všichni členové týmu a tak udržují všichni své kopie aktuální.

Systém Bazaar byl zamítnut hlavně proto, že s ním žádný člen týmu neměl zkušenosti, a také kvůli absenci hostingu pro hlavní úložiště.

Git, jehož popularita neustále roste, je založen primárně jako open-source nástroj pro open-source nástroje. V době začátků projektu Infinity sice nabízel dostatek hostingů pro hlavní úložiště (hlavně známý GitHub), ale vždy se jednalo o úložiště veřejné, tedy byly zdrojové kódy veřejně k dispozici.

Proto nakonec zvítězil Mercurial, který nabízel kvalitní hosting na serveru BitBucket.com, který byl zdarma a navíc uzavřený, tedy nikdo kromě týmu k úložišti přístup neměl. Navíc přinášel další výhody, jako například vestavěný Issue Tracker (viz kapitola 6.2 Nástroje pro týmovou spolupráci).

Výhodou Mercurialu byla také integrace v mnoha programátorských nástrojích, včetně IDE NetBeans, které tým využíval.

6.7 Nástroje pro analýzu a design

Bez ohledu na to, že většina současných metodik navrhuje krátké vývojové cykly, ať už se jedná o agilní techniky či Lean Development, na začátku musí do jisté míry proběhnout analytická fáze, připomínající staré techniky Waterfallu.

Není ale nutné (ani vhodné) provést analýzu a následně design celého vznikajícího systému, alespoň ne do takové hloubky, jak tomu bylo u vodopádového přístupu.

Místo toho stačí základní analýza – co bude produkt dělat? Co všechno zhruba má umět? Jaké technologie budou použity a kde? Jaký tým na to bude potřeba?

Odpovědi na tyto otázky se stanou nejen základem pro první cykly vývoje (kde se analýza právě vyvíjených částí samozřejmě ještě prohloubí), ale i prvky samotné strategie projektu. Souvislost s metodikou Lean Development je zřejmá – změna odpovědí na tyto otázky je velkou změnou strategie, pivotem, který má za následek významné změny ve vývoji, proto je nutné tyto otázky zodpovědět co nejdříve.

Základní otázky analýzy, tedy výběr týmu a technologií, zodpověděla minulá otázka. Dalším krokem je rozepsání budoucích funkcí vyvíjeného produktu, a obecně návrh jeho struktury.

Pro tento druh analýzy existuje mnoho dokumentačních technologií, které usnadňují a zpřehledňují zápis tohoto typu informací.

Nejznámějším jazykem pro popis vizualizaci, specifikaci a dokumentaci programových systémů je **UML** (Unified Modeling Language). Nabízí velké množství diagramů, které mohou zachytit prakticky všechny možné pohledy na vznikající systém.

Mezi nejznámější diagramy UML patří například diagram tříd, diagram aktivit, stavový diagram či například sekvenční diagram nebo diagram nasazení.

V rámci vývoje Infinity vzniklo takových diagramů hned několik, ale ukázalo se, že přinášejí týmu spolu s výhodami také velké množství nevýhod – vývoj v agilních cyklech často měnil strukturu některých částí aplikace tak často, že aktualizace zastaralých diagramů jen zdržovala vývoj, a neaktuální diagramy se postupně stávaly větším problémem než výhodou.

Například diagram tříd, zachycující strukturu databáze, se měnil tak často, a narostl do takových rozměrů, že se jeho využití stalo prakticky nemožným. Ukázalo se ale, že není vůbec potřeba – díky No-SQL databázi, kde neexistují relace, a veškerá propojení se provádí přes ID objektu a navíc díky aplikaci – hře, kde všechna spojení v databázi byla typu hráč-objekt (tedy realizována výhradně spojením přes ID hráče). Graf vztahů byl tedy zbytečný (všechny třídy byly jen spojeny s třídou Hráč) a jednotlivé třídy měnily své atributy příliš často, lépe tedy posloužil prostý seznam tříd, resp. kolekcí v tabulce.

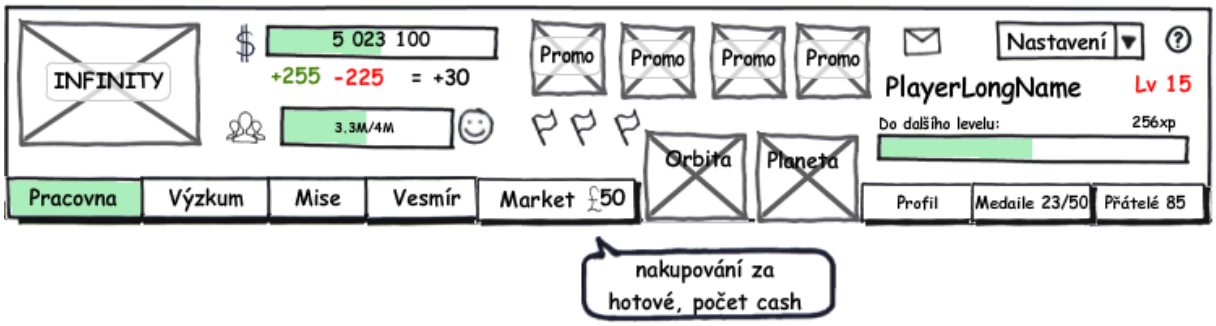
Tento příklad jasně ukazuje, že je vždy třeba využívat jen ty technologie, které jsou projektu přínosem, a ne jen slepě využívat všechny možnosti, které zvolené nástroje nabízí.

Diagramy, které byly nadále aktualizovány, se následně staly důležitou součástí vývojové dokumentace, která usnadní budoucí změny v systému a případné snazší pochopení systému pro nové programátory. Mezi těmito diagramy byl například diagram komunikace (klíčový pro aplikaci typu klient-server, v případě Infinity front-end – back-end) nebo diagram nasazení (důležitý z důvodu možného rozložení na více serverů, jak pro databázi, tak pro web-server).

V počáteční fázi analýzy tým využíval také některé nástroje na tvorbu tzv. **Mind maps**, tedy myšlenkových map. Pro PC existuje několik volně šiřitelných programů, které umožňují snadnou tvorbu těchto myšlenkových diagramů. Tento nástroj byl ale brzy opuštěn, protože

neumožňoval snadné sdílení – všichni členové týmu museli mít stejný nástroj a pro zasílání map museli využít jinou cestu (například e-mail), proto mapy brzy nahradily méně názorné, ale lépe šiřitelné a editovatelné nástroje (viz kapitola 6.2 Nástroje pro týmovou spolupráci).

Dalším nástrojem, který tým Infinity využil v pozdější fázi designu, byl nástroj na tvorbu tzv. **wireframes**, tedy náčrtů uživatelského rozhraní. Existuje velké množství programů, zabývajících se jejich tvorbou, tým Infinity využil nástroj Balsamiq Mockups.



Obrázek: Wireframe, první návrh hlavičky

Tyto náčrty sloužily například pro komunikaci s grafikem, nebo pro lepší představu, jak budou vypadat ty části programu, které měly vzniknout v pozdějších vývojových cyklech. Na konci vývoje tyto náčrty opět ztrácely smysl, protože během programování se uživatelské rozhraní často výrazně změnilo a proto se do finální dokumentace programu nedostaly, ale během vývoje byly neocenitelnou součástí návrhu.

6.8 Struktura aplikace

Posledním klíčovým rozhodnutím je stanovení struktury aplikace. V rozsáhlých systémech je vhodné využít některý z existujících návrhových vzorů – mezi nejznámější patří Model-View-Controller nebo jeho příbuzná varianta **Model-View-Presenter** (MVP), který využil i tým Infinity.

6.8.1 Model

Nejnižší vrstvou aplikace je Model, který zprostředkovává spojení s úložišti dat, jako je databáze nebo cache. Tato vrstva jako jediná smí mít znalosti o skutečné implementaci těchto úložišť.

V projektu Infinity ji tvořily jednotlivé třídy v jazyce PHP. Ty zařizovaly přístup do databáze Mongo či cache Memcached.

6.8.2 Presenter

Prostřední vrstvou tohoto návrhového vzoru je Presenter. Presentery v Infinity opět tvořily příslušné třídy v PHP. Zajišťovaly většinu aplikační logiky – řídily herní principy a jednotlivé algoritmy.

Tato vrstva neví, odkud pocházejí data, která využívá – Model sám rozhodne, jestli se načtou z cache nebo z databáze. Stejně tak presenter neví, jaká událost ho spustila – interakci s uživatelem má na starosti až následující vrstva.

6.8.3 View

View neboli pohled je nejvyšší vrstvou návrhového vzoru MVP. Obstarává všechno, co ve výsledné aplikaci vidí samotný uživatel – uživatelské rozhraní a obsluhu uživatelských událostí.

V Infinity pohled tvořil javascriptový front-end – řídil přepínání a vykreslování herních obrazovek, obsluhoval uživatelskou interakci a zasílal požadavky PHP back-endu, který za něj zařizoval veškerou aplikační logiku a perzistenci dat – pomocí obou výše zmíněných vrstev.

Toto rozdělení vrstev aplikace se osvědčilo – zejména oddělení pohledů do front-endu umožnilo samostatnou práci na obou vrstvách, v případě většího počtu programátorů by bylo možné rozdělit i programátory na práci na obou nižších vrstvách, v našem případě Modely i Presentery spravovali stejní programátoři.

7 Vývoj Infinity III – Minimální produkt

V této kapitole bude popsána další fáze vývoje Infinity, tedy fáze samotného programování. Popsán je vznik minimálního produktu a interakce s uživateli, ale i způsob programování a údržba a správa vznikající aplikace.

7.1 Vývoj aplikace

7.1.1 Agilní vývoj

Tým Infinity při programování používal metody agilního vývoje. Nesnažil se ale o naplnění některé z konkrétních implementací agilního přístupu (mezi které patří např. Scrum a další).

Základním agilním prvkem vývoje Infinity byly **týdenní iterace** – na začátku týdne se tým sešel, zkontroloval práci z minulého týdne a navrhl rozdělení úkolů na týden následující. Týdenní cykly pak zastřešovaly větší cykly, většinou dlouhé jeden či dva měsíce, které byly vždy zakončeny předem stanoveným **milníkem** – přesně specifikovaným stavem, který musí vyvíjená aplikace do té doby dosáhnout.

V souladu s metodikou Lean Development (opět bez její znalosti) ale tým tento přístup v průběhu vývoje opustil – postupně se (podle vlastních zkušeností jako prvních testerů produktu a později podle odezvy externích testerů) učil, které části aplikace by měly být vytvořeny co nejdříve, a které naopak mohou počkat.

7.1.2 Prototypy

Tým zvládl bez znalostí metodiky Lean Development vytvořit několik experimentů, které odpovídají definici a účelu minimálního produktu. Jednalo se vždy o prototypy, které pomáhaly zjistit, zda je daná technologie použitelná pro aktuální problém, či zda zvolené řešení skutečně funguje, nebo bude nutné vymyslet jiné. Tyto prototypy vznikly v rané fázi vývoje a nevyžadovaly zapojení testerů z řad uživatelů.

Typickým příkladem prvotních experimentů byl **prototyp vesmíru** pro druhou fázi hry. Podle prvního návrhu se mělo jednat o vesmír v podobě grafu – planety byly uzly grafu a hrany grafu měly reprezentovat vesmírné cesty mezi planetami. Prototyp však jasně ukázal, že v omezeném prostoru prohlížeče by takový vesmír nemohl být dostatečně přehledný pro

uživatele, ani snadný na vykreslení prohlížečem. Libovolné (šikmé) úsečky bylo možné vytvořit jen pomocí Canvasu, který byl příliš pomalý, pouhý posun v takovém vesmíru s desítkami až stovkami planet trval až jednotky sekund. Díky včasnému testování tedy mohl být opuštěn princip „vesmíru s hranami“ ve prospěch vesmíru v obyčejném dvojrozměrném euklidovském prostoru, kde se smí cestovat libovolně, ale i technologie Canvas, která se ukázala jako pomalá. Druhá generace tohoto prototypu již testovala vesmír bez hran, vykreslovaný pomocí pozicovaných HTML elementů a obrázků, a tento model se již ukázal jako dostatečně přehledný i výkonný.

Dalším prototypem byl experiment, týkající se **isometrického vykreslování budov** na planetě pouze za pomoci HTML elementů a CSS, pomocí kterého tým upřesnil, že tento systém nebude složitý na implementaci, pokud vykreslované budovy budou splňovat určitá rozměrová kritéria.

Posledním velkým prototypem byl **simulátor rozmísťování planet ve vesmíru**, který podle určeného algoritmu generoval souřadnice nových planet a vykresloval je do dvojrozměrného euklidovského prostoru). Ten objasnil, že zvolený systém pseudonáhodného rozmísťování nových hráčů do vesmíru funguje dostatečně spolehlivě – nové planety se objevují dostatečně blízko současným hráčům, v relativně pravidelných rozestupech, ale že tato pravidelnost není při bližším pohledu (který nabízí herní rozhraní) vůbec patrná.

Podle metodiky Lean Development by ale takových minimálních produktů mělo být mnohem víc – po jednom pro každou klíčovou vlastnost hry Infinity – ty by pak se zapojením ochotných testerů z řad early adopters pomohly včas odhalit nedostatky v designu hry, ale i priority hráčů – které funkce jsou pro ně klíčové, a které naopak mohl tým nechat na později či zcela vynechat.

Posledním minimálním produktem pak byla již téměř hotová aplikace.

7.2 Interakce s uživateli

7.2.1 Vývojář jako uživatel

Obecně se dá říct, že z obav o předčasné spuštění projektu (viz kapitola 8.1.1 a teoreticky kapitola 4.1.2) tým nevytvářel žádné „částečné“ experimenty pro uživatele,

a tyto rané testy prováděli pouze členové týmu. Ti se tak sami stali prvními uživateli nejen těchto prototypů, ale později i produktu samotného.

Pozdní zapojení externích testerů se samozřejmě ukázalo jako velký problém, ale samotný princip „Vývojář = uživatel“ se osvědčil. Členové týmu tak sami (často dříve než externí testeři) věděli o mnohých problémech aplikace, a také lépe dokázali komunikovat s uživateli aplikace – lépe chápali pohled samotných uživatelů.

V metodice Lean Development (a původní metodice Lean Manufacturing i systému TPS) tomuto přístupu odpovídá heslo „**Genči genbucu**“, tedy česky „Jdi a sám se podívej“, který vyzývá vývojáře (manažery, techniky, atd.), aby sami aktivně produkt zkusili, aby vnímali případné problémy na tom místě, kde se staly, ne jen z předzpracovaných zpráv atp.

7.2.2 Interakce s testery

V pozdní fázi vývoje, kdy byla výsledná aplikace téměř dokončená, bylo zahájeno testování s externími testery. Tým se obával předčasného vypuštění nedokončeného produktu, a tak se testování zpočátku týkalo jen předem vybraných uživatelů z řad „early adopters“, o kterých tým věděl, že budou tolerantní ke všem chybám, které může nedokončený produkt mít. Toto testování probíhalo dva týdny, během nichž tým dokončoval produkt dle vlastního plánu a zároveň reagoval na připomínky testerů.

Hlavním problémem tohoto přístupu nebyla uzavřenost testování, ale špatný výběr testerů – byli vybíráni zkušení testeři převážně z řad dalších programátorů. Jejich tolerance k chybám nevyvážila malou pestrost výběru – chyběli testeři z ostatních skupin uživatelů, hlavně méně zkušení uživatelé.

Tento problém tým vnímal, a proto brzy testování otevřel i pro ostatní uživatele, což již zajistilo dostatečně pestrou testovací skupinu.

Pro interakci s testery vznikla jednoduchá aplikace, de facto bug tracker, tedy aplikace pro správu chyb. Na rozdíl od volně dostupných bug trackerů, varianta vytvořená týmem Infinity byla napojena přímo na hru samotnou – uživatelé mohli nahlašovat problémy či například nesrozumitelné herní principy přímo prostřednictvím herního nástroje, bez nutnosti odcházet na jiný web a učit se používat nový nástroj. Tento přístup se osvědčil – své reakce poskytovali i uživatelé, kteří by externí nástroj (např. nástroj Bugzilla ad.) nedokázali či nechtěli použít.

7.3 Použité metriky

Metriky pro měření úspěšnosti projektu jsou jednou z mála kapitol vývoje, ve kterých se tým Infinity přiblížil metodice Lean Development jen minimálně. I proto v tomto případě neuspěl.

Využity byly pouze již zmíněné „vanity“ metriky (viz kapitola 4.2 Metriky), které pro měření úspěšnosti startupů nejsou vhodné.

Hlavní využitou metrikou byl **celkový počet uživatelů**, tedy počet uživatelů sociální sítě Facebook, kteří alespoň jednou spustili aplikaci Infinity. Tento počet neustále rostl, což ale neznamenalo, že byl projekt úspěšný – tato metrika nezohledňuje uživatele, kteří produkt pouze zkusili, a už se nikdy nevrátili.

Další variantou této metriky byl **denní počet nově příchozích uživatelů**. Tento počet byl téměř konstantní, nebo dokonce mírně rostoucí, i tak však započítával i nezajímavé uživatele, kteří produkt využili jen jednorázově.

Jedinou metrikou, která se blížila pojetí metrik dle metodiky Lean Development, bylo sledované procento aktivních uživatelů, kteří **postoupili z první fáze hry** do fáze druhé. Toto procento bylo relativně vysoké, což tým správně interpretoval jako úspěch.

Problémem však byla **analýza počtu aktivních hráčů**, tedy hráčů, kteří se do hry vraceli pravidelně. Tým nedokázal přesně definovat pojem „aktivní hráč“ – hráči se vraceli do hry mnohdy po týdnu neaktivity, nebo naopak hráli týden pravidelně, a poté skončili. Vzhledem ke krátké době měření tak nemohla být výběrová kritéria dostatečně ujasněno.

Z těchto důvodů ani kombinace posledních dvou metrik nebyla dostatečná. Posloužila by však jako dobrý základ funnel analysis, která by úspěšnost projektu Infinity měřila přesněji. Více o navrhovaných metrikách viz kapitola 8.2.

7.4 Vývoj mimo programování

Programování není jedinou činností, kterou se tým musel při vývoji Infinity zabývat. Ostatní nezbytné činnosti zabraly více než polovinu času (jedná se o odhad), potřebného pro samotný vývoj, tedy více než samotné programování. Následuje výčet těchto aktivit:

7.4.1 Návrh UI a UX

Předchozí kapitoly již zmiňovaly vytváření wireframes a návrh UI, tedy uživatelského rozhraní (anglicky User Interface). Součástí této kapitoly je ale i tzv. UX, tedy „User Experience“. Tato oblast se zabývá tím, jak uživatelé vnímají produkt prostřednictvím jeho UI. UX se snaží odpovídat na otázky typu:

- Co chceme, aby uživatelé na této stránce dělali?
- Jaký je hlavní ovládací prvek na této stránce?
- Které odkazy či tlačítka musí být výraznější, aby je uživatelé nepřehlédli, a která naopak můžeme znevýraznit či přesunout jinam?

či naopak

- Jaké možnosti uživatel od této stránky očekává?
- Kterou z aktivit na této stránce považuje uživatel za nejdůležitější?

Je nezbytné tyto otázky zodpovědět a uživatelské rozhraní navrhnout podle těchto odpovědí, jinak nemůže být UI pro uživatele dostatečně intuitivní a srozumitelné.

7.4.2 Správa serverů

Infinity potřebuje ke svému provozu hned několik aplikací, jak je zmíněno v kapitole 6. Webový server s PHP, databázový server, cache, to vše muselo být nainstalováno a udržováno nejen na hlavním produkčním serveru, ale také na lokálních serverech – každý člen týmu měl vlastní testovací server na svém počítači. Tyto servery musely být udržovány (např. aktualizovány) společně, aby prostředí jednotlivých serverů vždy odpovídalo produkčnímu serveru.

S větším počtem serverů také souvisí nutnost vytvoření testovacích dat – na produkčním serveru byli k dispozici „reální“ uživatelé (ze začátku jen tým Infinity, později i další testeři), na lokálních serverech ale bylo nutné další uživatele simulovat.

7.4.3 Vytvoření herních dat a principů

Kvalitní herní data a herní principy jsou nezbytné pro vznik kvalitní hry. Infinity jako strategická a budovatelská hra obsahuje desítky možností, které musely být dostatečně vyváženy, aby u hráčů hra mohla uspět.

Hra Infinity, jejíž statická herní data byla kvůli optimalizaci a možnosti verzování exportována přímo do zdrojového kódu, ve výsledku obsahuje více než 300 herních objektů

(budov, technologií, komponent pro stavbu kosmických lodí atd.), které mají vždy několik vlastností (10-30 dle typu), které bylo třeba kvantifikovat. Celkově herní data obsahují více než 300kB informací ve zdrojových kódech projektu Infinity.

Herní principy pak mj. zahrnují desítky vzorců (například vzorec pro výpočet délky vesmírné cesty v závislosti na motorech a hmotě lodi a vzdálenosti planet, ale i složitější vzorce).

Vlastnosti herních objektů a herní vzorce byly několikrát upravovány podle toho, jak se k nim stavěli uživatelé. Proběhlo několik měření typu „Které budovy hráči staví nejvíc“, podle kterých pak byl systém pozměněn tak, aby byly všechny alternativy použitelné a vyvážené. O nástrojích, sloužících pro tato měření, se zmiňuje kapitola 7.4.5.

7.4.4 Vytvoření textů

Hra Infinity obsahuje velké množství textů – od obsahu jednotlivých herních obrazovek/stránek, po názvy a popis jednotlivých herních objektů a jejich vlastností. Projekt Infinity navíc vznikal od začátku jako vícejazyčná aplikace, proto všechny texty vznikaly v češtině a angličtině, s vědomím, že další jazyky mohou být přidány později.

Celkem vzniklo více než 1300 textových řetězců, které v překladových souborech obou jazyků zabírají celkem 170kB.

7.4.5 Vývoj administračních nástrojů

Rozsáhlý projekt, jakým Infinity je, se neobejde bez velkého množství administračních nástrojů. Pro potřeby Infinity tak vznikla administrace, skládající se z cca 70 samostatných aplikací.

Některé byly pouze jednoduchými jednoúčelovými skripty, které sloužily například k naplnění databáze testovacími daty, které tým používal při vývoji na lokálních serverech. Jiné byly ale plnohodnotnými webovými aplikacemi.

Nejpoužívanější aplikací byl **překladač**, který sloužil k vytváření a správě herních textů. Tato aplikace umožňovala zapisování překladů pro všechny podporované jazyky a ukládání těchto překladů do souborů ve formátu JSON, které využívala samotná aplikace Infinity.

Mezi další aplikace patřilo například několik nástrojů pro **editaci herních dat** – například nástroj pro editaci herních budov sloužil k editaci všech budov ve hře a jejich vlastností v podobě tabulkového editoru, jeho zdrojový kód zabíral celkem více než 40kB.

Dále administrace obsahovala nástroj na **deployment**. Zařizovala stažení aktuálních zdrojových kódu ze serveru BitBucket, hlavního úložiště použitého verzovacího nástroje Mercurial.

Užitečný byl také nástroj na **přihlášení za libovolného uživatele**, který usnadnil testování na lokálním serveru s vygenerovanými uživateli, ale i řešení problémů reálných uživatelů na produkčním serveru. S tím souvisí i další nástroj, nástroj pro shromažďování, třídění a zpracování **odezvy od hráčů**.

Poslední zajímavou kategorií administračních nástrojů byly pak **statistické tabulky** – výpisy a statistiky čistých herních dat (například počty budov, které hráči postavili apod.). Z těchto nástrojů vycházely například analýzy využití herních dat, díky kterým bylo možné lépe vyvážit jednotlivá herní data.

7.4.6 Správa dokumentace

Velké projekty se neobejdou bez dokumentace. Základem dokumentace Infinity byly komentáře v kódu ve formátu Javadoc/PHPDoc. Tyto komentáře ale nebyly jedinou formou dokumentace – veškerá důležitá designová rozhodnutí byla zapisována do wiki projektu, postupně vznikala také finální uživatelská a vývojářská dokumentace. Součástí dokumentace bylo také několik sdílených dokumentů v aplikaci Google Docs, do kterých, jak bylo zmíněno dříve, bylo zapisováno např. rozdělení úkolů mezi jednotlivé vývojáře, ale také historie provedených herních změn či některé dočasné dokumenty, na kterých bylo třeba spolupracovat napříč týmem.

7.4.7 Komunikace s uživateli a propagace

V poslední fázi vývoje zabírala nejvíce času komunikace s uživateli. Nejednalo se jen o vyřizování uživatelských připomínek, získaných interním nástrojem pro uživatelskou odezvu, ale také odpovídání na tyto připomínky pomocí interní komunikace přímo uvnitř hry samotné.

Dále musela vzniknout informační webová stránka hry, obsahující obecné a propagační informace o hře, ale také přehledné tabulky herních dat, o které často žádali aktivní hráči.

Poslední částí propagace hry byla stránka hry uvnitř sociální sítě Facebook. Tam byly zveřejňovány informace o novinkách ve hře a také tam probíhala další komunikace s uživateli a fanoušky hry.

7.4.8 Testování

Testování bylo nedílnou součástí vývoje Infinity. Back-end aplikace byl testován pomocí unit testů pomocí technologie PHPUnit, front-end byl testován ručně. Všichni členové týmu také testovali přímo hraním samotné hry Infinity. Toto testování usnadňovaly další administrační nástroje, které umožňovaly rychlejší projití určitými fázemi hry, aby mohl člen týmu kdykoli hru resetovat, začít ji od začátku, a rychle odsimulovat průběh hry a postoupit právě do té fáze, kterou bylo třeba otestovat.

7.4.9 Komunikace s externím grafikem

Další čas zabírala komunikace s grafikem, který byl jediným externím spolupracovníkem týmu Infinity. Vždy bylo nutné přesně specifikovat potřebnou grafiku a obecně spolupracovat na návrhu grafického rozhraní atd.

Spolupráce s grafikem nebyla stvrzena písemnou smlouvou, což se ukázalo jako velký problém – grafik se velmi opožďoval s dodáním potřebné grafiky, což přispělo k neúspěchu projektu – tým se bál vypustit mezi uživatele hru bez grafiky, kde například budoucí isometrické obrázky budov nahrazovaly automaticky generované obrázky typu „krabice“.

8 Infinity – Návrh dalšího postupu

Tato kapitola obsahuje popis všech vážných prohřešků proti metodice Lean Development, které zapříčinily neúspěch projektu Infinity, a dále návrh dalšího postupu pro tento projekt, již zcela v souladu s metodikou Lean Development.

8.1 Chyby při vývoji Infinity

V průběhu vývoje projektu Infinity došlo k několika problémům, které tým nerozpoznal včas (či dokonce vůbec). V některých případech se jednalo o chybná designová rozhodnutí, jindy o přehlížení varovných náznaků.

V obou případech ale metodika Lean Development nabízí postupy, které právě těmto případům předcházejí – kdyby tedy tým od začátku tuto metodiku znal a aplikoval, mohl by těmto problémům snadno předejít.

8.1.1 Problémy s minimálním produktem

Tým Infinity potkal jeden z nejběžnějších problémů, týkajících se minimálního produktu – nepřiměřeně dlouho váhal s vydáním testovací verze, přestože byla určena jen pro vybrané testery z řad early adopters, kteří jsou vůči chybám prototypu velmi odolní a chápaví. Důvodem byly hlavně obavy z vypuštění programu, který by neobsahoval obě plánované fáze pro obě cílové skupiny, nebo kompletní grafiku.

Je zřejmé, že opačný přístup by byl výhodnější – dřívější spuštění testovací verze, obsahující jen jednu fázi hry (byť bez grafiky), kterou by mohli testeři podrobně otestovat a nebýt přitom ovlivněni fází druhou, by snáze a dříve objevilo nedostatky v hratelnosti počáteční fáze hry. Tento špatný postup byl klíčovou chybou, která zavinila celkový neúspěch projektu Infinity.

8.1.2 Cílové skupiny

Později v průběhu vývoje se ukázalo, že ve výběru cílových skupin došlo k další významné chybě. Sami členové vývojářského týmu Infinity spadali do druhé skupiny, tedy mezi pokročilé a zkušené hráče. První fáze hry ve snaze nebýt pouhou „hloupou kopií konkurence“ se proto stala v konečné podobě komplikovanější, než byla první cílová skupina (příležitostní hráči) ochotna akceptovat, což si tým neuvědomil dostatečně brzy. Ve spojení

s dalšími prohřešky proti technikám metodiky Lean Development (kterou tehdy tým Infinity neznal) toto vedlo k celkovému neúspěchu projektu.

Řešením by byl včasný Customer Segment Pivot, tedy změna cílové skupiny, v našem případě zaměření pouze na druhou cílovou skupinu. V tomto případě by byla první fáze hry upravena, aby odpovídala náročnějším požadavkům druhé skupiny.

Alternativou by bylo intenzivní testování s první uživatelskou skupinou, a odpovídající zjednodušení první fáze hry.

8.1.3 Virální šíření

Ohledně principu získávání nových uživatelů došlo k další chybě, nebo spíše podcenění a následnému technickému nedostatku.

V konkurenčních hrách na sociální síti Facebook je často „zvaní přátel“ nezbytnou součástí hry, ta až agresivně vyzývá hráče k rozesílání pozvánek, a často tím hráče úplně odradí a donutí k odchodu ze hry – hráč nemůže pokračovat, protože nebyl schopný do hry přivést vyžadovaný počet nových hráčů.

Proto tým Infinity vsadil na méně agresivní styl, který sice významně bonifikuje hráče, kteří mají ve hře své přátele, ale nijak je do této přátelské spolupráce nenutí, nevnucuje rozesílání pozvánek, jako ostatní hry, hru je možné hrát úspěšně bez jediného přítomného přítele. Tým ale podcenil význam tohoto nuceného zvaní na virální šíření, a tak získávání nových hráčů, které probíhalo hlavně na základě ústního šíření, nebylo tak rychlé, jak mohlo být.

Tento problém se ale netýká zvoleného přístupu jako takového, virální šíření i tak zůstává nejvhodnějším řešením pro aplikaci tohoto typu.

8.1.4 Spirála smrti velké dávky

Poslední chybou v projektu Infinity byl problém typu „Large-Batch spiral of death“, jak o něm mluví kapitola 2.2.2. Po obhájení projektu Infinity jako Softwarového projektu na MFF UK došlo ke špatnému rozhodnutí – namísto rychlého vypouštění dalších chystaných změn hry Infinity, tým rozhodl změny shromáždit do jedné velké aktualizace, na kterou bude předem uživatele připravovat a lákat. Propagační část tohoto rozhodnutí fungovala, vývojářská část ale narazila na výše zmíněný problém.

Velké množství změn s sebou neslo také velké množství nových chyb, které se těžko testovaly, jelikož bylo rozhodnuto nevypouštět tyto změny mezi uživatele, a testovat je pouze na lokálních serverech. S opravami těchto chyb se postupně přidávaly do velké aktualizace další jednotlivé změny a s nimi další nedostatky, až nakonec bylo vypuštění aktualizace odloženo na neurčito. Členové týmu, kteří po obhájení projektu během vleklého vývoje velké aktualizace ztratili motivaci dál pokračovat, tak projekt postupně opustili, a aktualizace zřejmě nebude nikdy dokončena.

8.1.5 Nevhodné metriky

Tento problém byl již zmíněn v kapitole 7.3 Použité metriky. Nevhodně zvolené metriky a analýzy nedokázaly včas odhalit některé z výše zmíněných nedostatků, jako byl například problém s cílovými skupinami (kapitola 8.1.2). Lépe zvolené metriky a analýzy, které navrhne kapitola 8.2, by přitom tyto problémy detekovaly dříve a tým by měl čas na ně reagovat.

8.1.6 Ověřování hypotéz

Tým Infinity bez znalosti metodiky Lean Development formuloval podrobně obě klíčové hypotézy, jak o nich mluví kapitola 3.1.2. Obhájil je ale pouze teoreticky, nepokusil se o jejich experimentální potvrzení. Takový experiment by mohl snadno odhalit designové chyby v první fázi hry, o kterých mluví kapitola 8.1.2.

8.2 Navrhované postupy

8.2.1 Metriky a analýzy

Cohort analysis

Základem nových metrik pro projekt Infinity by se měla stát kvalitně vypracovaná analýza kohort (viz kapitola 4.2.2).

Tým již disponoval všemi potřebnými daty – věděl přesně, který den se uživatelé přihlásili do hry poprvé, a tedy je mohl snadno rozdělit do kohort pro tento typ analýzy. Vzhledem k nízkým počtům těchto uživatelů na jedné straně a rychlému vývoji na straně druhé by bylo vhodné zvolit kompromisní velikost kohorty, zřejmě **1 týden** – tedy do jedné sledované skupiny přidělit hráče, kteří se poprvé přihlásili ve zvoleném týdnu.

Funnel analysis

Dále je nutné specifikovat jednotlivé vrstvy pro trychtýřovou analýzu (viz kapitola 4.2.1). Celek, tedy 100%, jsou **všichni uživatelé**, vybraní do kohorty podle minulého odstavce.

Dalším krokem je definice **aktivního uživatele**, která se stane druhou vrstvou „trychtýře“. Vzhledem k proměnlivému chování uživatelů (někteří se přihlásí podruhé až týden po prvním přihlášení a jiní naopak intenzivně hrají několik dní a pak odejdou) bude výhodné zvolit hned několik definic, a vytvořit vrstev více. Vzhledem k rychlému vývojovému cyklu ale není vhodné sledovat delší intervaly, než jeden týden – delší čekání (např. dva týdny či dokonce měsíc) na výsledky analýzy by vývoj zdržovalo.

Vhodnými vrstvami by byly tedy tyto: Počet hráčů, kteří se za první týden přihlásili **alespoň ve dvou dnech**, a Počet hráčů, kteří se za první týden přihlásili **v pěti a více dnech**. Je jasné, že první případ odfiltruje pouze hráče, kteří se po prvním přihlášení již nikdy nevrátili, započítá ale i ty hráče, kteří se vrátili jen jednou, bez ohledu na to, jestli to bylo hned druhý den, nebo až po týdnu. Tito hráči jsou jistě pro tým zajímaví – měli důvod se vrátit, ale zřejmě nebyl dostatečně silný, je zde tedy prostor pro zlepšení. Druhá zvolená kategorie, tedy pět a více dní v prvním týdnu, je naopak složená z aktivních hráčů, které hra potřebuje nejvíce, u nich je vysoká pravděpodobnost, že zůstanou i v dalších týdnech.

Tyto tři vrstvy (tedy všichni uživatelé, uživatelé s více přihlášeními, častí uživatelé) by stačily pro kvalitní základní analýzu počtu hráčů – procentuální velikosti druhé a třetí vrstvy by nahradili doposud používanou vanity-metriku Počet uživatelů.

V dlouhodobějším měřítku by tuto analýzu doplnily další dvě analýzy.

První z nich již tým používal – počet aktivních hráčů, kteří úspěšně **postoupí z první fáze** hry do druhé. Dříve byla problémem vágní definice aktivního hráče, nyní můžeme úspěšně použít třetí vrstvu minulé analýzy, tedy hráče, kteří hráli v prvním týdnu alespoň pět dní. Zjistíme tím procento hráčů, kteří hráli déle, ale nedokázali či nechtěli postoupit do druhé fáze. V tomto případě by bylo dále nutné rozlišit hráče, kteří se hrou skončili úplně (a na které je jistě nutné se zaměřit), a hráče, kteří chtějí zůstat v mírumilovné první herní fázi, a tedy zřejmě spadají do první cílové skupiny, jejich nevůle k přechodu do druhé fáze je pak na místě.

Data ke všem výše zmíněným metrikám již tým měl k dispozici, včetně počtu odehraných dní.

Poslední zajímavou analýzou by pak byla analýza růstu – vzhledem ke zvolenému virálnímu šíření tedy analýza **počtu přivedených přátel**. Zde by bylo na rozdíl od výše uvedených metrik nutné zavést nové získávání dat. V první řadě by bylo třeba zdokonalit systém pozvánek v rámci sociální sítě Facebook, aby se omezilo neměřitelné „ústní“ zvaní přátel, a aby byli hráči více motivováni využívat interní zvací systém, který měřit můžeme. Motivací by mohl být například bonifikační systém, zavádějící jednorázové herní odměny za úspěšné pozvání nového přítele do hry. Ve výsledku by tedy analýza měřila průměrný počet uživatelů, které stávající uživatel za celou dobu hraní pozve. Kvůli zjednodušení by se zřejmě měřil počet pozvaných přátel za první týden, a dále např. za první měsíc – můžeme předpokládat, že hráč, který intenzivně hraje celý měsíc, již pozval všechny přátele, které mohl, a tedy počet pozvánek po tomto datu bude zanedbatelný.

A/B testování

Dalším principem, který by tým Infinity musel zavést, je A/B testování. Vzhledem k existujícímu systému pro automatizaci deploymentu nových změn na produkční server by nebylo složité zavést takovou úpravu systému, aby nové změny dostávala vždy jen polovina nových uživatelů v daném týdnu. Pak by pomocí výše zmíněné analýzy kohort bylo možné měřit přesné účinky nově zavedených změn na chování hráčů – jestli hráči, kteří novou změnu k dispozici mají, hrají průměrně více dnů v prvním týdnu, než hráči bez této změny, byla jistě úspěšná. Pokud zůstanou výsledky měření pro obě skupiny stejné, změna byla zbytečná a nemá smysl v tomto směru pokračovat.

8.2.2 Kanban

Tým Infinity by měl zavést také systém Kanban, či jeho libovolnou odvozenou variantu. Cíl je zřejmý, minimalizovat počet zároveň vypuštěných změn do hry a lepší organizace pracovních úkolů.

Toto omezení by mělo za důsledek snáze dohledatelné a dokazatelné výsledky v souladu s principem 3A Metrik (viz kapitola 4.2.4) – tým by vždy přesně věděl, že chování uživatelů ve zvoleném týdnu zapříčinilo předem známé a malé množství nově nasazených změn produktu.

Nasazení principu Kanban by také zabránilo „spirále smrti“ (viz kapitola 8.1.4), protože by znemožnilo čekání na velké množství změn.

8.2.3 Experimentální ověření hypotéz

I když tento krok měl přijít hned v počátcích vývoje, během vzniku prvních minimálních produktů, i v pozdější fázi je možné experimentálně ověřit platnost základních dvou hypotéz, na kterých je postavena strategie produktu, tedy hodnotové hypotézy a hypotézy růstu.

V případě projektu Infinity by se jednalo hlavně o hypotézy, týkající se cílových skupin. Tým se dlouhodobě potýkal s nepřízní první cílové skupiny, tedy příležitostných hráčů. Měl by tedy podstoupit rozsáhlou analýzu jejich chování, pokusit se zodpovědět například následující otázky:

- Vadí první cílové skupině konkrétní herní princip, či obecně zaměření celé hry?
- Pokud je problémů v tomto směru více, které z nich jsou nejzávažnější?
- Pomohla by prostá změna UI, která by pokročilejší herní prvky před první cílovou skupinou ukryla?

Finální otázkou této analýzy by bylo rozhodnutí, zda bude pro tým výhodnější (levnější, rychlejší, méně náročné…) předělat hru tak, aby si získala přízeň i této skupiny, nebo zda bude lepší změnit strategii v tomto bodě a první cílovou skupinu zcela opustit. Tato změna by se ale týkala ale i samotné vize produktu – první fáze hry by tak víceméně ztratila svůj smysl. Bylo by tak nutné produkt radikálně změnit – první fázi zcela odstranit, nebo alespoň výrazně přizpůsobit potřebám druhé cílové skupiny, pro kterou neměla být původně určena.

Prostředkem pro tuto analýzu by mohlo být například rozsáhlé A/B testování, spojené s intenzivním dotazováním uživatelů z první cílové skupiny.

9 Souhrn – Vývoj krok za krokem

Tato kapitola shrne všechny fáze vývoje webové aplikace bez zadavatele, tak jak je absolvoval tým Infinity při vývoji webové hry Infinity, resp. jak je měl absolvovat, pokud by postupoval plně v souladu s metodikou Lean Development.

Bude také zdůrazněno, ve kterých bodech se tým Infinity od této metodiky nejvíce odklonil a kde naopak (bez znalosti této metodiky) postupoval takřka přesně podle jejích pravidel.

Kapitola tak může sloužit jako návod či checklist, obsahující všechny kroky, které musí tým při vývoji webového startupu splnit, aby mohl nejen dokončit vývoj aplikace, ale také uspět jako startup.

9.1 Fáze 1 – Analýza

9.1.1 Vize

V první fázi je nezbytné stanovit vizi projektu, tedy základní nosné myšlenky a nápady, které projekt definují.

O této fázi mluví teoreticky kapitola 3.1.1, prakticky o vizi Infinity pak kapitola 5.1. Tým Infinity svou vizi podrobně definoval.

9.1.2 Strategie

Jako druhý krok při vývoji webového startupu vzniká strategie projektu, jako souhrn základních designových a technologických rozhodnutí.

Do této fáze spadá základní analýza a design, podobně jako u běžného vývoje aplikace se známým zadavatelem.

Technologie

V rámci volby strategie je třeba zvolit technologie, na kterých bude aplikace postavena.

U webové aplikace se jedná o **databázový server**, **webový server**, **serverový skriptovací jazyk**, **značkovací a skriptovací jazyky** pro front-end a v neposlední řadě také o základní **strukturu aplikace**, která definuje, jak tyto komponenty spolupracují.

Je vhodné již v této chvíli zvolit také ostatní technologie pro vývoj – vývojové prostředí, verzovací systém nebo nástroje pro týmovou spolupráci.

Informace o volbě technologií pro webovou aplikaci a konkrétně pro projekt Infinity obsahuje kapitola 6.

Tým

V této fázi je nutné brát v úvahu také schopnosti týmu – a buď postavit tým podle zvolených technologií, či naopak při výběru technologií zohlednit schopnosti již vybraných členů vývojářského týmu – tým Infinity zvolil druhou možnost.

Je také dobré předem počítat s faktem, že se žádný ze členů týmu neomezí pouze na programování, ale bude se účastnit i ostatních částí vývoje startupu – komunikace s ostatními vývojáři, komunikace s uživateli, měření a analýza chování uživatelů atp.

Základní hypotézy

Strategii projektu je vhodné postavit na několika základních hypotézách, o kterých mluví kapitola 3.1.2. Těmito hypotézami jsou Hodnotová hypotéza a Hypotéza růstu.

Tým Infinity tyto hypotézy formuloval a alespoň teoreticky obhájil, viz kapitola 5.2.

9.2 Fáze 2 – Programování

9.2.1 Plán vývoje

Pro samotné programování je vhodné postupovat podle zásad agilního vývoje.

Je nutné vědět, že se metodika Lean Development s agilním vývojem nevylučuje – pouze upravuje některé jeho části, které pro vývoj bez zadavatele nejsou vhodné (například zrušením milníků vývoje).

Některými fázemi vývoje (např. organizace samotného programování) se naopak Lean Development nezabývá vůbec a nechává tak tyto části pouze na úvaze týmu – v těchto případech je vhodné nasadit právě prvky agilních metodik, alespoň pokud nejsou v přímém rozporu se zásadami metodiky Lean Development.

Tým Infinity v této fázi postupoval podle vlastní interpretace agilního přístupu, viz kapitola 7.1.1.

9.2.2 Minimální produkty

V první fázi samotné programovací fáze je třeba experimentálně ověřit pravdivost obou hypotéz, formulovaných v předchozí fázi. K tomu slouží minimální produkty, jejichž účel je jakýmkoli způsobem ukázat uživatelům záměr projektu a sledovat, zda jejich reakce odpovídají našim hypotézám. V případě negativních reakcí je třeba tyto hypotézy odpovídajícím způsobem upravit. Více o minimálních produktech viz kapitola 4.1.

Výsledný produkt se tak může v prvotní fázi výrazně změnit. Tato změna – takzvaný Pivot – ale předchází neúspěchu, který by nastal, pokud by tým dokončil vývoj produktu, aniž by byly pravdivé jeho základní hypotézy. Více o případných zásadních změnách viz kapitola 4.3.

Tým Infinity neověřil dostatečně pravdivost svých hypotéz a tím částečně zapříčinil pozdější neúspěch projektu, viz kapitoly 7.1.2 nebo 8.1.1.

9.2.3 Návrh uživatelského rozhraní a UX

Návrh uživatelského rozhraní je nedílnou součástí vývoje a vyžaduje často zvláštní nástroje, viz kapitola 6.7.

Vyžaduje nejen grafické cítění a případnou komunikaci s grafikem, ale také předvídání chování uživatelů a jejich požadavků, viz UX v kapitole 7.1.4.

9.2.4 Data a texty

Nedílnou součástí webové aplikace je často velké množství textů (mnohdy ve více jazycích) a také různých dat – např. v případě webové hry se jedná o data, popisující herní objekty. Dalším typem dat jsou pak data testovací, která v prvotních fázích vývoje nahrazují data, která později budou generovat samotní uživatelé.

Více o datech a textech projektu Infinity viz kapitoly 7.4.3 a 7.4.4.

9.3 Fáze 3 – Interakce

9.3.1 Učení

Hlavním cílem týmu při vývoji aplikace bez zadavatele je učení – tým musí postupně zjistit, jak přesně má jeho výsledný produkt vypadat, aby uspěl. V předchozí fázi vývoje za tímto účelem vytvářel minimální produkty, v této fázi musí poslední z minimálních produktů (základ budoucího finálního produktu) upravovat podle reakcí uživatelů.

Tým tedy musí s uživateli intenzivně komunikovat, reagovat na jejich reakce na produkt.

V projektu Infinity probíhalo dostatečné množství komunikace s uživateli, kteří byli ochotni produkt využívat, tým ale měl iniciovat komunikaci i s ostatními, méně zkušenými a komunikativními uživateli, aby získal více informací o jejich základních problémech.

9.3.2 Metriky

Učení, zmíněné v předchozím kroku, zahrnuje také zkoumání chování uživatelů prostřednictvím analýzy statistických dat, získaných během provozu aplikace.

K tomuto účelu je nutné zavést několik metrik, které budou odpovídat konkrétnímu projektu a jeho záměrům a principům. Více o výběru metrik viz kapitola 4.2.

Projekt Infinity nedokázal správně stanovit své metriky, což bylo jedním z důvodů celkového neúspěchu projektu. Navrhované metriky již v souladu s metodikou Lean Development pro projekt Infinity popisuje kapitola 8.2.1.

9.3.3 Administrace

Správa aplikace, ale i jejích dat a textů, nebo měření statistik podle předchozí kapitoly, to vše vyžaduje dostatek nástrojů pro administraci aplikace.

Projekt Infinity zahrnoval rozsáhlou administraci - více o administraci aplikace Infinity viz kapitola 7.4.5.

10 Závěr

Při vývoji startupů, tedy aplikací bez zadavatele, hrozí velké riziko neúspěchu – vývojářský tým předem nezná cílovou skupinu svého vznikajícího produktu, neví, jestli produkt u uživatelů uspěje. Stávající metodiky, zaměřené primárně na vývoj s předem známým zadavatelem, cílovým prostředím i uživateli, nejsou pro vývoj bez zadavatele dostatečné.

Metodika Lean Development navrhuje postupy a principy, mající za účel snížit riziko z neúspěchu při vývoji startupů. Její metody umožňují objevit dostatečně včas většinu známých problémů, typických pro startupy, a reagovat na ně. Reakcí může být jejich oprava, nebo v případě závažnějších problémů včasné ukončením projektu, které ušetří zdroje, které by jinak byly vynaloženy na vývoj produktu, odsouzeného k neúspěchu.

Hlavním znakem metodiky Lean Development je učení – vývojový tým se učí, jaká je cílová skupina jeho produktu, jak je třeba produkt upravit, aby u cílové skupiny uspěl, učí se, jak na produktu vydělat. Cílem je minimalizace „odpadu“ – částí produktu, které nepřispějí k jeho úspěchu, ke zlepšení přijetí mezi cílovou skupinou, ke zvýšení zisku. Mezi základní principy patří častá interakce s uživateli, nové metriky pro měření úspěšnosti startupu, a metody, zajišťující včasnou změnu přístupu v případě, že měření a komunikace s uživateli ukáže, že stávající přístup nevede ke zlepšení přijetí produktu.

Přístupy Lean Development se nevylučují se stávajícími metodikami, staví jen další vrstvu, která pomáhá eliminovat jejich nedostatky při vývoji bez zadavatele – je tedy možné (a vhodné) kombinovat principy metodiky Lean Development například s agilním vývojem, jak to ukázal tým Infinity na případě svého projektu.

Tým vyvíjel svou webovou aplikaci – hru Infinity – bez znalosti metodiky Lean Development, přesto se však v některých klíčových rozhodnutích principům této metodiky velmi blížil. Bohužel se však nedokázal vyhnout několika základním problémům, typickým pro webové aplikace bez zadavatele a startupy obecně, které měly za následek celkový neúspěch projektu. Tato práce však ukázala, že s dostatečnou znalostí metodiky Lean Development by tým mohl tyto problémy napravit, nebo alespoň zjistit jejich existenci dostatečně brzy, a zásadně změnit podle těchto zjištění celý produkt tak, aby se těmto problémům zcela vyhnul.

Práce dále definuje kompletní seznam činností, které musí tým, vyvíjející webovou aplikaci bez zadavatele podstoupit – počínaje stanovením cílů projektu, analýzou a rozhodnutími ze sféry softwarového inženýrství a konče samotným programováním a měřením úspěšnosti projektu. Tyto kroky ilustruje na příkladu projektu Infinity, včetně výběru technologií, aktuálních pro rok 2012 a nejbližší budoucnost.

Pro projekt Infinity byly také navrženy konkrétní kroky, které umožní pokračování projektu ze současného stavu. Tyto kroky reagují na všechny zjištěné problémy a v souladu s metodikou Lean Development určují jejich řešení.

Tým Infinity se tedy během případného dalšího vývoje dokáže vyvarovat nejzávažnějších chyb a zvládne také napravit chyby minulé.

11 Citovaná literatura

1. **Ries, Eric.** *The lean startup: how today's entrepreneurs use continuous innovation to create radically successful businesses.* New York : Crown Business, 2011. ISBN 978-0-307-88789-4.

2. Lean manufacturing. *Wikipedia.* [Online] [Citace: 6. červen 2012.] http://en.wikipedia.org/wiki/Lean_manufacturing.

3. Informační společnost v číslech 2012. *Český statistický úřad.* [Online] Český statistický úřad, 27. 3 2012. [Citace: 20. 6 2012.] http://www.czso.cz/csu/2012edicniplan.nsf/p/9705-12.

4. "The World in 2011: ITC Facts and Figures". *ITU.* [Online] 2011 . http://www.itu.int/ITU-D/ict/facts/2011/material/ICTFactsFigures2011.pdf.

5. Web Server. *Wikipedia.* [Online] Wikimedia Foundation. [Citace: 2. 7 2012.] http://en.wikipedia.org/wiki/Web_server.

6. Nginx: the High-Performance Web Server and Reverse Proxy. *Linux Journal.* [Online] 1. 7 2008. [Citace: 2. 7 2012.] http://www.linuxjournal.com/article/10108.

7. Usage of server-side programming languages for websites. *W3Techs - Web Technology Surveys.* [Online] [Citace: 2. 7 2012.] http://w3techs.com/technologies/overview/programming_language/all.

8. Canvas Browser Support. *Caniuse.com.* [Online] [Citace: 12. 7 2012.] http://caniuse.com/#search=canvas.

9. Javascript Frameworks. *Wappalyzer.* [Online] [Citace: 10. 7 2012.] http://wappalyzer.com/categories/javascript-frameworks.

10. *Projekt Infinity - Vývojová dokumentace.* 2012.

11. *Symposium on advanced programming methods for digital computers.* **Benington, Herbert D.** Washington, D.C., USA : Office of Naval Research, Dept. of the Navy, 1956. OCLC 10794738.

12. **Larman, Craig a Basili, Victor R.** Iterative and Incremental Development: A Brief History. *Computer, ISSN 0018-9162.* 2003.

13. **Larman, Craig.** *Agile and Iterative Development: A Manager's Guide.* místo neznámé : Addison-Wesley, 2004. ISBN 978-0-13-111155-4.