Sem vložte zadání Vaší práce.

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

Bakalářská práce

Studie metodik řízení softwarového vývoje

Daniel Schmidt

Vedoucí práce: Ing. Jiří Hunka

10. května 2017

Poděkování

Ačkoli jsem autorem bakalářské práce i zadání, studie by nemohla vzniknout nebýt několika klíčových lidí. Předně chci poděkovat vedoucímu Jiřímu Hunkovi za to, že mě vedl a pomohl mi udržet směr i v obtížnějších chvílích. Velký dík patří Marku Polčákovi, Václavu Podlipnému a Ondřeji Hrabalovi z Quanti, kteří mi významně pomohli zkontaktovat řadu společností, a tím uskutečnit výzkum samotný – pro mě pravou podstatu práce. Dále děkuji zástupcům managementu jednotlivých firem, jenž si na mě vyhradili čas a byli ochotni o problematice diskutovat. Jmenovitě Davidu Černému ze Senmanu, Dominiku Veselému z Ackee, Tomáši Hercegovi a Ladislavu Šestákovi z Riganti, Janu Randovi a Vladimíru Beranovi z Etnetery, Pavlu Rybovi z 2N, Davidu Kozelkovi, Janu Pacovskému a Jiřímu Turkovi z Adastry, Lukáši Honzákovi, Michaele Eichinger Žekovové a Luboši Laginovi z Avastu. Děkuji i všem ostatním, kteří se na výzkumu jakkoli podíleli. Děkuji Tomáši Nováčkovi, protože si práci jako první celou přečetl a pomohl mi s korekturou. Na závěr děkuji dvěma lidem, kteří tu pro mě vždy jsou, Lucii a Janu Schmidtovým, mým rodičům.

Prohlášení

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

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

V Praze dne 10. května 2017 .....

Odkaz na tuto práci

Schmidt, Daniel. *Studie metodik řízení softwarového vývoje.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Bakalářská práce je studií metodik pro řízení projektů při vývoji aplikací. Účelem je zjistit, jaké metodiky IT firmy reálně používají, jaký je kontext a motivace jejich použití. Do výzkumu zahrnuji sedm firem různých velikostí. Pro srovnání zařazuji metodiku vyučovanou během bakalářského studia na FIT ČVUT v Praze. Data pro výzkum získávám polo-strukturovaným rozhovorem se zástupci managementu jednotlivých firem. Verifikaci dat jsem provedl pomocí volitelného pozorování v příslušných společnostech. Jednotlivé metodiky popisuji a následně vyhodnocuji na základě zvolených srovnávacích kritérií a provedení simulace průběhu fiktivního projektu dle dané metodiky. Zhodnocení doplňuji o další úhly pohledu, včetně výtažku esenciálních praktik moderního přístupu k řízení projektů.

Klíčová slova metodiky softwarového vývoje, životní cyklus softwarového vývoje, projektový management, porovnání metodik, nejlepší praktiky

Abstract

The bachelor thesis researches software development processes with the purpose of determining which project management frameworks are used by IT companies in a particular context during real projects. The research group

consists of seven companies with different backgrounds and a methodology taught at the bachelor's program at FIT CTU in Prague. Qualitative research methods are used for data collection. I had used a semi-structured interview for the data collection. The data were later validated by an optional participant observation. The respective methodologies are described and evaluated using selected measurements and a simulation of their usage during a fictional project. Finally, I sum up the best practices of the modern project management approach.

Keywords software development frameworks, software development life cycle, project management, methodologies comparison, the best practices

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Žijeme v době neklesajícího boomu vývoje aplikací, kde IT řešení pronikla do takřka všech odvětví lidské působnosti. Zvyšují se očekávání uživatelů, vývojové týmy zahrnují mezioborové odborníky a dodávané systémy jsou komplexnější a sofistikovanější, než kdy dříve.

Tato skutečnost klade nároky na řízení a organizaci vývojového procesu, na jehož konci stojí právě ony aplikace. Nehmatatelná podstata aplikací umožňuje vést projekty stylem, které jsou v ostatních oborech vázaných fyzikou nemyslitelné. V počátcích nového tisíciletí zažil softwarový vývoj významnou revoluci v podobě agilních metodik. Mimo samotných metodik se rovněž rozvíjí nové technologie, jež umožňují zefektivnění činností softwarového inženýrství.

Metodik existuje celé množství a orientace v nich je obtížná. Stejně jako v jiných oblastech, i zde není nic černobílé a v rozdílných situacích dodávají nejlepší výsledky metodiky různé. V rámci studie si kladu za cíl zmapovat reálné využití metodik v IT společnostech zabývajících se softwarovým vývojem. Vybrané společnosti se kontextuálně liší, což dělá práci ojedinělou. Proces výzkumu je založen na kvalitativních metodách. Práce jde do hloubky u užšího vzorku společností, kde hledá důvody použití konkrétních principů a postupů. Výsledný popis metodik je detailní a rozšířen o kontextuální souvislosti.

Zajímavost dodává studii fakt, že do výzkumu zařazuji metodiku vyučovanou během bakalářského studia FIT ČVUT v Praze. Tento aspekt vytváří kontrast mezi akademickým a v praxi používaným přístupem. Jedním z přínosů pro čtenáře studenty je rozdíl mezi tím, s čím se potkají během studia a posléze po nástupu do praxe.

Praktická část studie podrobuje metodiky simulaci na ukázkovém projektu středního rozsahu, kde zkoumám dopady externích vlivů na průběh projektu. V závěru studie přináším hodnocení metodik z různých úhlů pohledu, včetně závěrečného vyzdvihnutí stěžejních principů řízení moderního vývoje.

Kapitola 1

Softwarový vývoj

Úvodní kapitola shrnuje základní teorii softwarového inženýrství. Čerpám z velké části z literatury Software Engineering: A Practitioner's Approach od Roger Pressmana [4] a Software Engineering od Iana Sommerville [5].

1.1 Rozdělení software

Softwarové inženýrství je specifické vůči ostatním technickým disciplínám jako je strojírenství nebo stavitelství. Software je totiž abstraktní a nehmotný. Není vázán vlastnostmi materiálů nebo řízen zákony fyziky. Absence přírodních limitů do jisté míry zjednodušuje jeho vývoj. Na druhou stranu, díky nepřítomnosti fyzikálního omezení, software dokáže rychle nabrat na komplexnosti vyúsťující v náročnost pochopení jeho fungování a nákladnost na úpravu.

Rozlišujeme dvě kategorie softwaru:

- *Produktový* software je samostatný produkt vyvinutý developerskou společností a prodáván na trhu. Produkt si může opatřit kdokoli, kdo má na jeho pořízení prostředky.
- *Zakázkový* software je vyvíjen pro konkrétního klienta. Řešení je šito na míru jeho potřebám.

Patrným a důležitým rozdílem je strana, která specifikuje systém. V produktovém případě se jedná o developerskou společnost, kdežto u zakázkového vývoje je to zákazník, kdo určuje podobu systému. Tento fakt hraje významnou roli v přístupu během vývoje softwaru. V zakázkovém vývoji figurují minimálně dvě strany, které spolu musí kolaborovat. Je nutné podotknout, že poslední dobou rozdíly splývají. Vývoj produktového softwaru je často ovlivněn zpětnou vazbou trhu a klíčových odběratelů [6]. Na druhou stranu, trendem zakázkového softwaru je přizpůsobení již existujícího řešení, což lze pozorovat u dodavatelů robustních informačních systémů.

Existují i dvě základní rozdělení z hlediska financování vývoje:

- *Fixed time fixed price* (FTFP). Projekt disponuje pevným rozpočtem, stanoveným dnem dodání a má definovaný rozsah požadavků. Zákazník ví přesně co a za kolik dostane. Na druhou stranu, dodavatel nemusí podrobně vykazovat odpracovanou práci a často optimalizuje náklady využitím subdodavatelů. Kvůli trendu podhodnocování projektů plynoucí z kuželu nejistoty je metoda obecně výhodnější pro zákazníka.

- *Time & Material* (T&M). Dodavatel je placen podle vykázaného času na základě dohodnuté hodinové mzdy. Přístup dává zákazníkovi větší volnost v definici a změně požadavků. Nicméně zákazník nemá ukotvený cíl ani termín dodání, což pro něj představuje určitou míru nejistoty. Metoda je náročnější na management a kolaboraci stran. Snadno se může stát, že projekt překročí plánovaný rozpočet zákazníka. Z pravidla výhodnější pro dodavatele ze stejného důvodu jako předchozí metoda.

Typ kontraktu hraje významnou roli při projektovém řízení. Ukazuje se, že flexibilnějším metodikám vývoje více prospívá financování T&M.

1.2 Přístupy vývoje softwaru

Slovníková definice metodiky je: *„teoreticko-praktické schéma určující postup provádění odborné činnosti. Vychází z vědeckého poznání a empirie, přesně vymezuje jednotlivé postupy pro výkon dané činnosti.“* [7]

Během vývoje softwaru probíhají tzv. aktivity softwarového inženýrství. Jde o činnosti, které probíhají za účelem vytvoření software. Příkladem může být specifikace požadavků, implementace nebo testování. Každá literatura dělí formálně aktivity trochu jinak. Ve výsledku se jedná o to samé. V kontextu studie dává metodika tyto aktivity do souvislostí. Jedná se o framework, který říká v jakém pořadí a míře jednotlivé aktivity probíhají. Z hlediska filosofie můžeme metodiky rozdělit do třech základních kategorií.

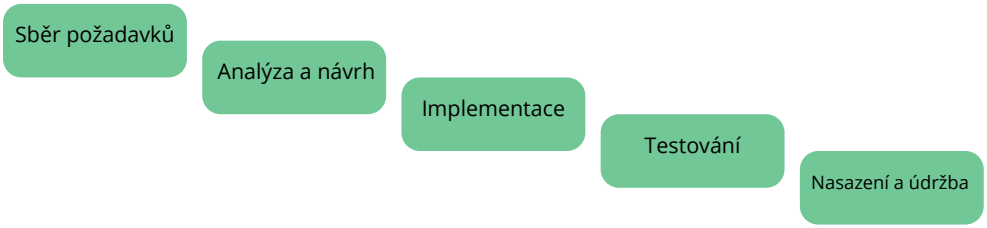
1.2.1 Vodopád

Historicky nejstarším přístupem je vodopád. Jedná se o tradiční, jedno cyklický přístup, v rámci kterého probíhají jednotlivé aktivity po sobě a bez možnosti návratu. Každá aktivita je tedy provedena právě jednou. Sled aktivit znázorňuje obrázek 1.1. Aktivity jsou jasně odděleny, z pravidla formou dokumentu (zadávací specifikace, architektura řešení, předávací protokol...). Z definice vodopádu plyne:

- Specifikace požadavků musí být kompletní před začátkem vývoje a nemění se.

- Nasazení probíhá po dokončení celé implementace a testování.

- Průběh projektu je plánovaný. Již po analýze a návrhu se ví, jak bude probíhat vývoj, testování i nasazení.



Obrázek 1.1: Vodopád

Vodopád je terčem kritiky na základě uvedených bodů. Zprvu je obtížné dodržet lineární přístup aktivit. Zákazník často nemá na začátku jasno, co přesně chce, a proto nedokáže kompletně specifikovat chování systému. Model je nepružný vůči změnám po ukončení sběru a analýzy. Představení funkčního řešení zákazníkovi na konci vývoje je rizikové, protože až v této fázi uvidí co dostane, jak je s tím spokojen a zda byla naplněna jeho očekávání. Nadto se může ukázat, že zákazník tou dobou řešení již nepotřebuje nebo potřebuje odlišnou funkcionalitu, ať už z jeho vlastní vůle nebo vzhledem k externím okolnostem (legislativa, situace na trhu, technologický posun...).

I přes tyto nedostatky je vodopád stále používaný. Vodopád je velmi efektivní například u krátkých projektů nebo projektů s jasnou představou. Dále skutečnost, že každá aktivita probíhá pouze jednou, razantně snižuje náklady. V neposlední řadě se jedná o základní přístup k vývoji softwaru, ze kterého ostatní do větší či menší míry vychází.

1.2.2 Iterativní přístup

Důvodem vzniku iterativního přístupu je jeden z nedostatků vodopádu - nasazení do provozu až na konci projektu. Výsledkem je přístup založený na více cyklech. Počet cyklů se různí, typicky se jedná o jednotky. Na iterativní přístup se dá rovněž pohlížet jako na sérii menších vodopádů.

Myšlenkou je rozložit systém na dílčí části, které lze dodat postupně. Každá část je realizována v samostatné iteraci, kde výstup se označuje jako softwarový inkrement. V rámci iterace probíhají aktivity v obdobném duchu jako u vodopádu s rozdílem, že rozsah je omezen na danou část. Velká část pojmenovaných, popsaných metodik staví na iterativním přístupu. Nejznámější je Unified Process.

Stejně jako vodopád, i u iterativního přístupu se narazilo na problémy. Ukazuje se, že pozdější iterace v době vývoje naráží na konstrukční problémy. Často se stávající kód musí upravovat, aby šla implementovat nová funkcionalita. Kód je náchylnější opomenutí best practices a coding standards. Tento

aspekt není problémem u vodopádu, protože počáteční návrh architektury řešení zastřešuje všechny požadavky a ne jen podmnožinu. Z tohoto důvodu se často první iterace obětuje více analýze a návrhu řešení z pohledu celku, avšak jen do omezené míry. [8]

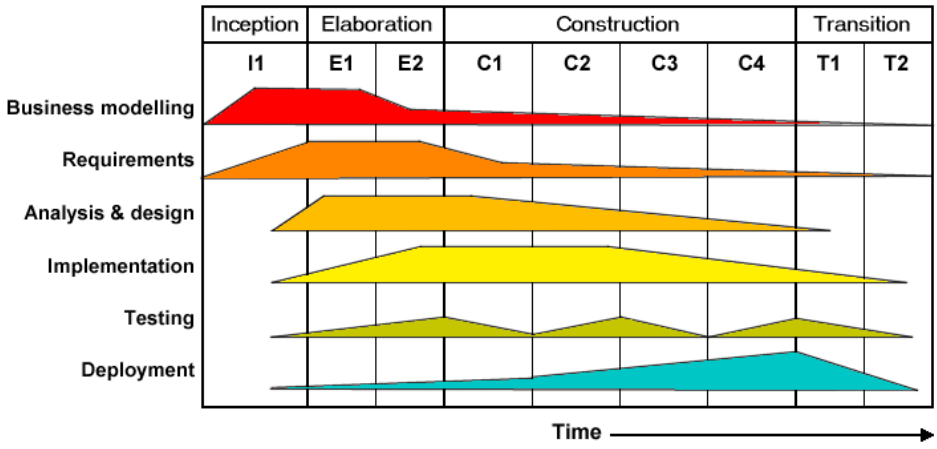
Opakování aktivit jako je testování nebo nasazení s sebou nese zvýšení nákladů. Kromě vynaloženého času je problémem i častější součinnost ze strany zákazníka. Potíž u testování je, že při akceptaci se netestuje pouze přidaná funkčnost softwarového inkrementu, ale i nezměněné chování dosavadního systému. Při n iteracích je tedy výstup první iterace otestován n krát, výstup druhé $n-1$ krát atp. [8]

1.2.2.1 Unified Process

Unified process (UP) představuje robustní metodiku softwarového vývoje. Často je známá i pod názvem Rational Unified Process. Rational je divize IBM, která Unified Process detailně zdokumentovala a začala používat v praxi. [1] Na rozdíl od ostatních metodik rozděluje proces na fáze a aktivity. Obrázek 1.2 ukazuje distribuce aktivit během jednotlivých fází.

Metodika je silně spjata s modelovacím jazykem UML. Výstupy z jednotlivých etap jsou dokumentovány pomocí specifických UML diagramů. Konstruované řešení z dokumentace vychází a při změně muzí být obě verze konzistentní.

Funkční požadavky jsou v UP vymezeny případy užití (use case). Jedná se o krok za krokem popsané scénáře toho, jak uživatel interaguje se systémem. Use case jednoduchou formou vyjasňuje očekávané chování systému ze strany zákazníka.



Obrázek 1.2: Unified Process

UP definuje čtyři projektové fáze. Fáze se dívají na projekt v čase z businessového pohledu. Každá fáze má jasný cíl.

- *Inception* představuje místo pro identifikování všech zainteresovaných stran, jejich očekávání a přínos systému. Dodavatel se seznamuje s doménou zákazníka a jeho fungováním. Vytváří se hrubý plán řešení.

- *Elaboration* je fáze plánování. Rozšiřuje výstupy z předešlé fáze, jde se více do hloubky. Cílem je zmapovat všechny případy užití, navrhnout architektonický model řešení a vytvořit plán vývoje spolu s vyhodnocením rizik.

- *Construction* zastřešuje implementaci řešení, včetně integrace dílčích částí, podle návrhu z předcházející fáze. Během této fáze je systém vyvinut a interně otestován.

- *Transition* popisuje přesun vyvinutého řešení do reálného používání. Na začátku probíhá akceptační testování odvozené z popsaných use cases. Po nasazení do produkce následuje monitoring.

Aktivity jsou oproti fázím statické disciplíny, které naplňují jednotlivé fáze v různém poměru. Tabulka 1.1 ukazuje rozdělení primárních a sekundárních aktivit.

Tabulka 1.1: Aktivity UP [1]

Primární	Sekundární
Business modelování	Konfgurační řízení
Sběr požadavků	Projektové řízení
Analýza a návrh	Řízení prostředí
Implementace	
Testování	
Nasazení	

Primární aktivity se přímo podílí na vývoji systému, vytváří hodnotu. Naopak sekundární disciplíny slouží podpůrně primárním. Jak je vidět z obrázku 1.2, aktivity probíhají kontinuelně, přeš všechny fáze na rozdíl od vodopádového přístupu. Sekundární aktivity oproti primárním probíhají stabilněji a bez větších výkyvů.

Nejdůležitějším rysem UP je rozdělení na fáze a aktivity. Dále uvědomění, že přechod do produkce je nedílnou součástí procesů a nejedná se pouze o formalitu, kterou si zákazník nějak vyřeší. Aktivity pomáhají naplnit cíle jednotlivých fází. Nejsou jednoznačně spjaty s určitou fází, namísto toho probíhají neustále v různé intenzitě.

1.2.3 Agilní metodiky

Slovo agilní je trendem dnešní doby. Každý je agilní, nebo to o sobě alespoň tvrdí. Agilita znamená reagovat na změny a být jim otevřen. Jedná se o změnu jakéhokoli typu. Změna v týmu, změna požadavků, změna technologie. Agilní tým si uvědomuje, že základem úspěšného projektu jsou schopní jedinci a jejich kolaborace. [9]

Roku 2001 podepsalo 17 odborníků z IT branže Manifest Agilního softwarového vývoje [10], který praví:

Objevujeme lepší způsoby vývoje software tím, že jej tvoříme a pomáháme při jeho tvorbě ostatním. Při této práci jsme dospěli k těmto hodnotám:

- *Jednotlivci a interakce* před procesy a nástroji
- *Fungující software* před vyčerpávající dokumentací
- *Spolupráce se zákazníkem* před vyjednáváním o smlouvě
- *Reagování na změny* před dodržováním plánu

Jakkoliv jsou body napravo hodnotné, bodů nalevo si ceníme více.

Autoři výroku, nazývající se Agilní aliance, rovněž definovali 12 principů agilního vývoje:

1. Naší nejvyšší prioritou je vyhovět zákazníkovi časným a průběžným dodáváním hodnotného softwaru.
2. Vítáme změny v požadavcích, a to i v pozdějších fázích vývoje. Agilní procesy podporují změny vedoucí ke zvýšení konkurenceschopnosti zákazníka.
3. Dodáváme fungující software v intervalech týdnů až měsíců, s preferencí kratší periody.
4. Lidé z byznysu a vývoje musí spolupracovat denně po celou dobu projektu.
5. Budujeme projekty kolem motivovaných jednotlivců. Vytváříme jim prostředí, podporujeme jejich potřeby a důvěřujeme, že odvedou dobrou práci.
6. Nejúčinnějším a nejefektnějším způsobem sdělování informací vývojovému týmu z vnějšku i uvnitř něj je osobní konverzace.
7. Hlavním měřítkem pokroku je fungující software.
8. Agilní procesy podporují udržitelný rozvoj. Sponzoři, vývojáři i uživatelé by měli být schopni udržet stálé tempo trvale.

9. Agilitu zvyšuje neustálá pozornost věnovaná technické výjimečnosti a dobrému designu.

10. Jednoduchost–umění maximalizovat množství nevykonané práce–je klíčová.

11. Nejlepší architektury, požadavky a návrhy vzejdou ze samo-organizujících se týmů.

12. Tým se pravidelně zamýšlí nad tím, jak se stát efektivnějším, a následně koriguje a přizpůsobuje své chování a zvyklosti.

Obecně vzato, agilní metodiky ctí agilní manifest a jeho 12 principů. Každá metodika naplňuje principy v různém poměru a odlišnou formou. V porovnání s iterativním přístupem, je agilní vývoj extrémnější. Cykly jsou kratší a je jich mnohem více, často se nazývají sprinty. Tvoří se méně dokumentace, soustředění je na vývoj, release a zpětnou vazbu. Agilních metodik je pojmenováno několik. Mezi ty nejznámější patří Scrum.

Agilní metodiky byly vynalezeny pro kompaktní týmy, sídlící na jednom místě s možností operativní komunikace. Z povahy agility je přístup využíván především pro systémy malého a středního rozsahu, potažmo jako nástroj ve fázi údržby. Nabízí se snaha zrobustnit agilní přístup i pro velké systémy, na jejichž vývoji se podílí velké množství týmů i v geograficky odlišných lokalitách.

Škálovatelnost agilních metodik pro velké systémy je aktuální téma. Sommerville [5] věří, že agilní metodiky se musí u velkých projektů adaptovat v následujících bodech:

1. U velkých systémů se nelze zaměřit pouze na samotný kód. Je důležitý návrh architektury a dokumentace kritických částí systémů, které jsou sdíleny více aplikacemi.

2. Návrh komunikačního rozhraní pro virtuální týmy. Dnešní doba nabízí možnosti jako video konference, telefonické hovory nebo online chatovací aplikace pro usnadnění komunikace.

3. CI při každé změně je nereálný v situaci, kdy na kódu a rozhraní jedné aplikace závisí další systémy. Častý build je nicméně jedním ze základů agility, a proto je zde prostor pro nové nástroje konfiguračního řízení podporující mezi-aplikační synchronizaci.

S podobným smýšlením jsem se setkal i při rozhovoru se seniorním managing consultantem ze společnosti Adastra, Janem Pacovským, podle něho: *„Agilní metodiky fungují dobře při implementaci monolitických systémů, nedokážu si je představit v situacích, kdy je potřeba integrovat více systémů dohromady.“* [11]

Lindvall [12] problematiku posouvá o úroveň výše a zaměřuje se na problematiku použití ve velkých společnostech jako takových, kde argumentuje:

1. Pro společnosti může být agilita kulturním šokem po dlouhé historii vedení projektů konzervativním způsobem. Analogicky, projektoví manažeři (PM) zvyklí na tradiční styl řízení se mohou zdráhat nejasnému cílovému výstupu, jenž s sebou agilita přináší.

2. Velké společnosti mohou mít zavedené vnitro-podnikové procesy, které jsou v rozporu s agilním přístupem. Příkladem takových procesů je změnové řízení nebo zajištění kvality. Stejný problém může nastat v nucení používání určitých softwarových nástrojů napříč projekty.

3. Agilní metodiky kladou vysoké nároky na senioritu a přesah lidí. Velké společnosti typicky disponují řadou úzce profilovaných specialistů spolu s mladými, ale zatím nezkušenými, talenty.

Co platilo jako negativum u iterativního přístupu v porovnání s vodopádovým, platí u agilního dvojnásob. Na druhou stranu se jedná o směr vpřed, neb poslední výzkumy ukazují, že při uplatnění agilních a iterativních metodik dodávají projektové týmy řešení více odpovídající představám zákazníka. [13]

1.2.3.1 Scrum

V dnešní době se Scrum řadí mezi nejznámější agilní metodiky. Pojmenování vychází z rugbyové akce restartování hry po malém prohřešku zvaného scrummage. Metodika byla prvně presentována v polovině devadesátých let Kenem Schwaberem a Jeffem Sutherlandem. Stejně jako ostatní agilní metodiky, i Scrum ctí agilní manifest. Popis metodiky čerpá z oficiálnách zdrojů tzv. Scrum Alliance [14].

Metodika definuje role dohromady tvořící Scrum team. Každá role hraje určitou úlohu a řídí se danými pravidly během vývoje.

- *Product Owner* (PO) je osoba odpovědná za požadavky. Jedná o člověka vyhraněného zákazníkem, který je plně k dispozici vývojovému týmu. PO formuluje úkoly a určuje jejich prioritu, čímž formuje podobu vyvíjeného produktu.

- *Scrum Master* (SM) je obdoba projektového manažera. Stejně jako PM vede tým a odstraňuje překážky vývojovému týmu. Hlavním úkolem SM je však dohlížet na to, zda jsou dodržovány všechny praktiky Scrumu.

- *Development team* (DevTeam) je skupina vývojářů realizující požadavky. Tým je samoorganizující a vývoj je plně v jejich režii. Každý člen týmu nese označení vývojář, neexistují výjimky nebo podtýmy podle technologií. Na vývojáře jsou kladeny nároky seniority, protože společně požadavky analyzují, odhadují, realizují i testují.

Události ve Scrumu mají fixní délku, jasný průběh i frekvenci opakování. Základní jednotkou Scrumu je sprint. Jedná se o iteraci v délce řádově 1-4 týdny, na jejímž konci je funkční softwarový inkrement. Délka sprintu i náplň je pevně dána. Sprint se skládá, mimo samotného vývoje, ze čtyř formálních událostí:

- *Plánování* sprintu provádí celý Scrum tým v délce maximálně jednoho dne. Přesunou se úkoly z product backlogu do sprint backlogu. Určí se cíl sprintu (tzv. goal), což je typicky větší funkcionalita skládající se z dílčích úkolů ve sprintlogu. Cíl sprintu pomáhá vést Develpoment team. Koncem plánování je odpovězení otázek: Co se dodá na konci sprintu a co je pro to potřeba udělat.

- *Denní scrum meeting* (DSM) je aktivita probíhající každý den. Jedná se 15min schůzku DevTeamu organizovanou SM, kde se plánuje práce na další den. Každý člen odpoví na otázky:

  - Co jsem udělal včera pro dosažení cíle sprintu?
  - Co udělám dnes pro dosažení cíle sprintu?
  - Jaké vidím překážky bránící ve vývoji mě nebo někomu jinému?

- *Review* se koná na konci sprintu, v rámci kterého se posuzuje nově vyvinutá funkcionalita (inkrement) předáná zákazníkovi. Mimo to se řeší nově přidané požadavky v product backlogu.

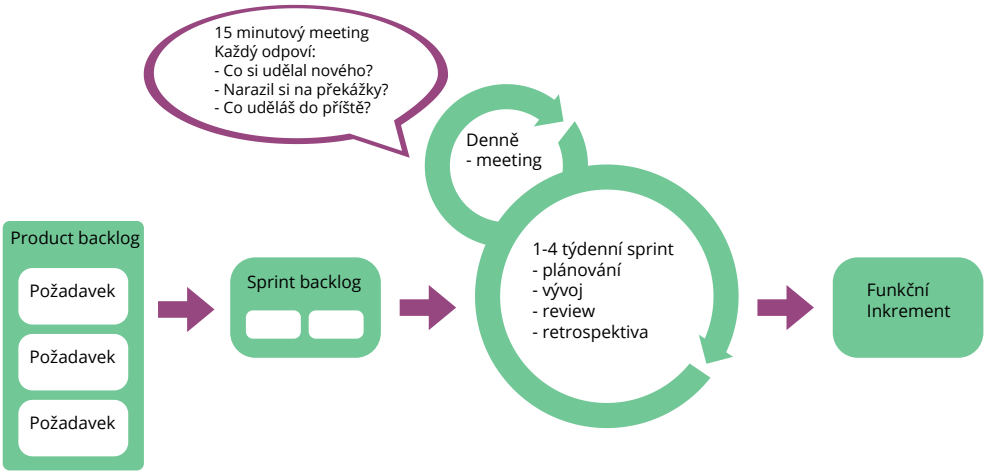
- *Retrospektiva* následuje po Review. Jejím účelem je diskutovat průběh posledního sprintu, přijít s návrhy na vylepšení včetně postupu jejich integrace. Předmětem vylepšení jsou vztahy v týmu, procesy i nástroje. Výstupem je plán vylepšení flow pro další sprint.

Artefakty representují práci a hodnotu ve Scrumu. Scrum rozeznává tři artefakty:

- Základním artefaktem je *Product backlog*, list všech požadavků k vyvinutí. Za Product backlog je zodpovědný PO, jenž do backlogu přidává úkoly a řadí je podle priority. Naprosto všechny požadavky prochází product backlogem, ať už se jedná o nové požadavky, úpravy stávající funkcionality, rozšíření či opravy. Každý úkol má svůj popis, časový odhad a prioritu.

- *Sprint backlog* je list úkolů pro daný sprint. Tvoří se při plánování sprintu a plní se úkoly z Product backlogu podle priority.

- *Inkrement* je verze systému ke konci sprintu. Jedná se o sumu všech dokončených úkolů během sprintu aktuálního a předchozích.



Obrázek 1.3: Scrum

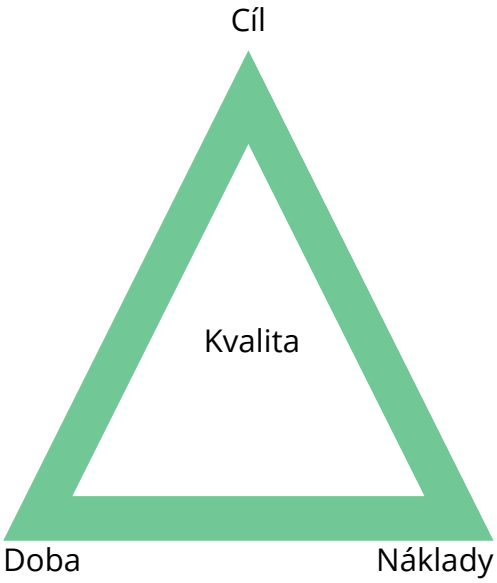
Obrázková representace 1.3 dává popsané body do souvislostí. Jak je vidět, Scrum dává agilitě nástroje pro transparentnost, revizi a adaptibilitu, což jsou mimo jiné i základní pilíře této metodiky. Podle Beedla, *„Scrum umožňuje vývojovým týmům pracovat úspěšně ve světe, kde eliminace nejistoty je nemožná.“* [15]

1.3 Srovnávací kritéria metodik

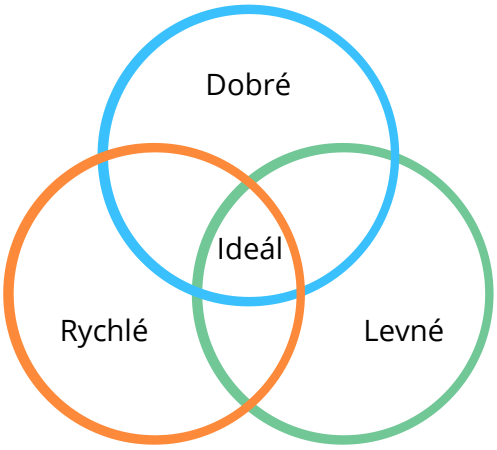
Pro hodnocení metodik v rámci studie volím projektový trojimperativ [16]. Jedná se o fundamentální pohled na úspěšnost projektu ze tří, respektive čtyř pohledů. Těmi pohledy jsou:

- *Cíl.* Projekt končí dodáním výstupů v požadovaném rozsahu. Cíl chceme maximalizovat.
- *Doba.* Čas jsou peníze. Projekty mají čas dodání. Dobu chceme minimalizovat.
- *Náklady.* Součástí projektu je omezený rozpočet. Náklady chceme minimalizovat.
- *Kvalita.* Dodávka splňuje kvalitativní normy zákazníka. Často se opomíjí a jako první se vypouští. Kvalitu chceme obecně maximalizovat.

Ilustorváno na obrázku 1.4, cípy pomyslného trojúhelníku tvoří první tři pohledy, tedy cíl, náklady a dobu, kde kvalita figuruje jako centrální prvek. Ideální stav je maximalizace cíle v požadované kvalitě za minimalizace doby a nákladů. Snahou projektu je naplnit trojimperativ, dosáhnout ideálního stavu. Alternativní pohled na imperativ ukazuje obrázek 1.5



Obrázek 1.4: Trojimperativ



Obrázek 1.5: Kruhový trojimperativ

Komplikací je, že cípy trojúhelníku jdou proti sobě. Vylepšení jednoho kritéria má za následek zhoršení minimálně jednoho z ostatních. Ideální stav je v praxi nedosažitelný. Projekty reálně končí v rovnováze všech cípů nebo naplnění dvou cípů ze tří.

V rámci studie budu metodiky hodnotit podle cípů trojimperativu, kde kvalitu zahrnu pod cíl.

- *Cíl* představuje dodání toho, co klient chce. Uvažuji, že maximalizaci cíle ovlivňuje:

1. Čas a forma specifikace požadavků.
2. Frekvence nasazení a zpětné vazby od zákazníka.
3. Frekvence komunikace se zákazníkem.
4. Reakce na změny a jejich zapracování.

- *Doba* trvání kritické cesty, jenž beru jako čas od započetí projektu po okamžik, kdy se do provozu dostane kompletní verze systému. Simulované metodiky budu měřit Ganttovými diagramy.
- *Náklady* jsou komplexní proměnná zahrnující několik faktorů.

  1. Nároky na velikost týmu a týmové role, odvíjeno od rovného počtu vývojářů pro každou metodiku.
  2. Nároky na vykonání aktivit softwarového inženýrství. Příkladem může být opakované testování nebo údržba dokumentace.

  Náklady přepočítám pomocí počtu a vytíženosti zdrojů.

Prioritu volím konzervativně, stejnou pro všechna tři kritéria. Při reálných projektech záleží na kombinaci několika faktorů. [17] Interní projekty zpravidla maximalizují cíl na úkor času a nákladů. U projektů pro externí klienty záleží na mnohem více proměnných. Jedná o jednorázový projekt, nebo dlouhodobou spolupráci? Jak je dodání kritické pro zákazníka z businessového pohledu? Jedná se o nový systém nebo náhradu stávajícího? Je projekt součástí většího projektu, kde náš termín dodání ovlivňuje jeho harmonogram? Vzhledem k této subjektivitě a zásadnosti každého kritéria, neupřednostňuji žádné kritérium před ostatními.

Kapitola 2

Výzkum

V následující kapitole je popsán plán, metody a výstupy provedeného výzkumu během studie. Pro načerpání teorie a volby optimálního výzkumného procesu jsem čerpal z literatury profesora Brymana z Oxfordu, Social Research Methods. [2] Výzkum slouží pro získání odpovědí na základní otázky, které jsou podkladem pro následnou praktickou část práce. Těmito otázkami jsou:

- Jaká je používaná metodika pro řízení softwarového vývoje?
- Jaká je motivace pro její použití?
- V jakém kontextu se metodika používá?

Podstata výzkumu je tedy zodpovězení otázek: co, proč a kdy.

2.1 Vzorkování

Důležitou roli ve výzkumu hraje počátečních zdroj dat. Zdroje dělíme na:

- Primární: data získaná samotným výzkumníkem
- Sekundární: již existující data (statistiky, předešlé výzkumy . . .) [18]

Způsob volby primárního zdroje dat, v mém případě společností, se nazývá vzorkování (angl. sampling). Vzhledem k požadavkům zadání práce jsem využil tzv. Theoretical sampling. Podstata tkví v tom, že vzorky nejsou vybírány náhodně, nýbrž podle kritérií.

Požadavky na společnost jsou v případě studie velikost firmy a nutnost zabývat se softwarovým vývojem. Na rozdíl od účelového vzorkování se tato metoda používá iterativně. Firmy jsou oslovovány ve vlnách dokud není dostatek dat pro pokračování ve výzkumu. Tento stav se nazývá saturace.

2.2 Výzkumný plán

Výzkumný plán je šablona specifikující filosofii průběhu výzkumu. Vzhledem k zadání a stanoveným cílům studie jsem zvolil porovnávací plán. Logika porovnání ukazuje, že dokážeme lépe pochopit danou oblast, pokud vzájemně porovnáme několik rozdílných případů. Pro zmapování jednotlivých případů se typicky využívá principů průřezové studie. Ta se vyznačuje tím, že data jsou sbírána v jednom časovém okamžiku.

V okolnostech této studie je případem myšlena společnost, respektive vysokoškolský předmět. Principy průřezové studie umožňují získat data od subjektů při jedné návštěvě. Porovnávací plán je poté nadstavbou korespondující s myšlenkou práce.

2.3 Výzkumné metody

Výzkumnou metodou se rozumí samotný způsob získávání dat. Existují dva typy přístupu, kvantitativní a kvalitativní. Kvantitativní přístup je založen na číslech a velkém vzorku dat. Data sesbíraná kvantitativní metodou jsou strukturovaná a snadno počítačově zpracovatelná. Odpovědi připravuje výzkumník a výstupy jsou zobecněné. Příkladem kvantitativních metod je dotazník nebo strukturované formy interview či pozorování.

Naproti tomu kvalitativní výzkum preferuje slova místo čísel. Zkoumá hlubší význam a procesy chování v daném kontextu. Výzkumník je více zapojen do samotného sběru dat a odpovědi tvoří tázaný účastník. Mezi metody kvalitativního výzkumu se řadí nestrukturované formy interview a pozorování nebo zasedání cílových skupin. Tabulka 2.1 uceluje rozdíly mezi jednotlivými přístupy.

Tabulka 2.1: Porovnání kvalitativního a kvantitativního přístupu [2]

Aspekt	Kvantitativní	Kvalitativní
přístup	čísla	slova
úhel pohledu	vědce	účastníka
účast vědce	odstíněn	zapojen
forma dat	strukturovaná	nestrukturovaná
povaha dat	mnoho, spolehlivá	bohatá, do hloubky
výstup	zobecnění	kontext

Z povahy studie a velikosti dotazovacího vzorku volím kvalitativní výzkumné metody. Jmenovitě dvoustupňový proces ve formě polo-strukturovaného interview v kombinaci s následným pozorováním. Kvalitativní přístup je vhodný pro zjištění skutečného fungování řízení projektů ve společnostech, včetně zjištění specifik a motivace konkrétního použití. Daní za bohatá data je vyšší náročnost získávání a zpracování dat.

2.3.1 Interview

Rozhovor jsem zvolil jako primární zdroj dat, protože oproti ostatním metodám kvalitativního výzkumu je nejméně náročný na čas a lidskou součinnost. Existují tři typy rozhovoru porovnané v tabulce 2.2.

Tabulka 2.2: Typy interview [2]

Typ	Povaha	Průběh
Strukturované	kvantitativní	uzavřené otázky
Polo-strukturované	kvalitativní	osnova
Nestrukturované	kvalitativní	rámcové téma

Jako vhodný typ se ukázalo být polo-strukturované interview, kde výzkumník má sérii obecnějších otázek s podotázkami. Tato osnova vytváří rámec rozhovoru, kterého se výzkumník drží. Účastník na dotazy odpovídá volně a formulace otázek se může lišit od sezení k sezení. Metoda tedy zaručuje přirozené pokrytí potřebných bodů v kvalitativní formě.

2.3.1.1 Osnova interview

- představení sebe a práce
- kontext firmy
  - struktura
  - kategorizace
  - projektové portfolio a typy odběratelů
  - vize
- popis používané metodiky pro řízení projektů
- aktivity softwarového inženýrství
  - řízení
  - sběr požadavků
  - analýza
  - návrh
  - implementace
  - QA
  - konfigurační řízení
  - řízení prostředí
  - dokumentace

- diskuse popsaného řešení
  - motivace
  - výhody řešení
  - prostory ke zlepšení
- akceptace poznámek
- zeptání se na pozorování

Pokrytí uvedené osnovy zabírá 60 až 90 minut. V rámci přípravy jsem provedl zkušební rozhovor s manažerem ze společnosti Quanti [19], kde pracuji. Z důvodu objektivity jsem se rozhodl Quanti do výzkumu nezapojit. Požadavkem na účastníka z mé strany je manažerská pozice v dané firmě. Jako pozornost jsem vždy přinesl manažerovi lahev moravského vína. Podobnou osnovu jsem posílal vždy předem účastníkovi, aby věděl, čeho se bude rozhovor týkat.

Důležité jsou poslední dva body. Z rozhovoru jsem si dělal textové poznámky na papír. Poznámky jsem s účastníkem na konci interview prošel, abych měl jistotu, že jsem jednak nic nevynechal a jednak vše správně pochopil. S dotazem na pozorování přicházím na konci interview, protože tím zvyšuji šance na kladnou odpověď oproti scénáři, kdy o to požádám rovnou spolu s interview. V tom případě bych mohl být odmítnut nejen ohledně pozorování, ale i na samotný rozhovor. [20]

2.3.2 Pozorování

Pozorování je druhým krokem sběru dat a slouží zejména jako verifikace výstupu z interview. Dalším účelem pozorování je lepší pochopení metodiky popsané během rozhovoru. Cílem výzkumného procesu je totiž zjistit, jak řízení průběhu probíhá ve skutečnosti namísto toho, jak má modelově vypadat.

Jedná se o nepovinnou část, neb řadě firem je nepříjemné absolvovat samotný rozhovor, natož přistoupit na pozorování, ačkoli se jedná o akademickou práci bez účelu poškození jména společnosti či prozrazení know-how.

U společností, které nepodstoupily pozorování nelze říci, zda vývoj softwaru ve skutečnosti probíhá tak, jak bylo v rozhovoru popsáno, a tudíž jsou zjištěná data neověřená a méně věrohodná. Navzdory okolnostem jsou výstupy zařazeny do praktické části studie. Výzkum je nastaven tak, že data z interview pro další práci stačí. Pozorování je pouze ověřuje a dává je do lepších souvislostí.

Stejně jako rozhovor, i pozorování má svoji kvantitativní a kvalitativní formu. Shrnutí základního rozdělení je v tabulce 2.3. Strukturované pozorování je kvantitativní forma. Definuje striktní pravidla, časový harmongram a formát výstupu, podle kterého jsou jednotlivé subjekty pozorovány. Analogicky ke strukturovanému rozhovoru, výzkumník zaujímá rovný přístup ke

všem. Naopak účastněné pozorování představuje formu kvalitativní. Výzkumník je součástí kolektivu po daný časový úsek. Sleduje, co se děje, o čem se lidé baví a čerpá z individuálních interview. Podstatou přístupu je mimo vypozorování vzorce chování i zjištění jeho důvodu.

Tabulka 2.3: Typy pozorování [2]

Typ	Povaha	Průběh
Strukturované	kvantitativní	pevná pravidla a harmonogram
Účastněné	kvalitativní	přirozený, součást kolektivu

Při realizaci pozorování jsem do dané firmy přišel a strávil v ní celý den. Myšlenkou je projít celodenním cyklem, jako stálý zaměstnanec za účelem maximálního vcítění se a nasání atmosféry. Každá návštěva tedy trvala přibližně 8 hodin. Jeden den jsem zvolil jako kompromis mezi přínosem a realizovatelností. Z povahy kvalitativního výzkumu jsem využil principů účastněného pozorování. Minimální osnova pokrývala:

- doptání se na nejasnosti
- verifikace informací z rozhovoru
- získání praktické představy
  - metodiky řízení projektu
  - podoby dokumentů

Řada firem mi byla nápomocná a uspořádala mi program, kde jsem se připojil k PM, jeho týmu a absolvoval s nimi stand up, interní porady či schůzku se zákazníkem. V případě, že program byl volnější nebo nastala mezera, využil jsem situace k individuálnímu výzkumu. Seznamoval jsem se s lidmi napříč odděleními a povídal si s nimi o jejich úloze a fungování v rámci projektu i firmy. Často jsem se ptal i na věci, které už jsem věděl z rozhovoru nebo jež jsem slyšel od někoho jiného. Verifikace.

Pozorování předčilo má očekávání. Ukázalo se jako pomocnější, než jsem očekával. Bohatství a kvalita výstupů je na zcela jiné úrovni mezi společnostmi, kde návštěva proběhla a neproběhla. Ohledně právního rámce, pouze jedna společnost požadovala podepsat mlčenlivost. Přisuzuji to tomu, že prvotní rozhovor je fáze dělící firmy na ty, které se chtějí, respektive nechtějí otevřeně bavit o problematice řízení projektů.

2.4 Oslovené společnosti

2.4.1 Kategorizace

Do výzkumu se zapojují malé, střední a velké společnosti. České právo nicméně tyto pojmy nezná. [21] Pro účely studie se inspiruji definicí pojmů podle

CzechInvestu - státní příspěvkové organizace pro podporu podnikání a investic. [22] Kritérii jsou počet zaměstnanců a roční obrat. V situacích, kdy na základě jednotlivých kritérií spadá společnost do různých kategorií, je stěžejní velikost firmy, tedy počet zaměstnanců.

Tabulka 2.4: Kategorizace, data: CzechInvest [3]

Kategorie	Počet zaměstnanců	Obrat (mil. Kč)
malá	< 50	< 250
střední	< 250	< 1 250
velká	>= 250	>= 1 250

2.4.2 Společnosti

Jak již bylo zmíněno, při výběru společností jsem použil metodu Theoretical sampling. Postupně jsem sehnal kontakt na desítky firem z nichž jsem vybíral postupně jednak podle kategorie společnosti a jednak podle vlastního zájmu. Kompletní seznam potenciálních účastníků studie je v příloze B.1. Společnosti, které se staly součástí výzkumu, jsou uvedeny v tabulce 2.5.

Výběr obsahuje pestrou škálu firem nejen co do velikosti. Avast a 2N se zabývají výhradně produktovým vývojem. Specialitou Riganti je britská a zámořská klientela. Řada firem vyvíjí i pro státní sektor. Rozmanitost výzkumného vzorku snižuje neobjektivitu a zkreslení dat. Vysoká kvalita vstupních dat je předpokladem pro relevantní výsledky studie.

Tabulka 2.5: Oslovené společnosti

Společnost	Velikost	Obrat (mil. Kč)	Kategorie	Interview	Pozorování
Senman	6	4	malá	ano	ano
Ackee	30	30	malá	ano	ne
Riganti	45	30	malá	ano	ano
Etnetera	150	180	střední	ano	ano
2N	230	800	střední	ano	ne
Adastra	400	750	velká	ano	ano
Avast	2100	17 500	velká	ano	ano

Data o velikosti a obratu společností odpovídají stavu při první fázi výzkumu, tedy rozhovorech, jenž probíhaly na přelomu let 2016-17. Účelem těchto dat je pouze rozčlenění společností do kategorií.

2.5 Metodiky

Kapitola popisuje výstupy z výzkumné části. U každého participanta je popsána nejen metodika použitá při řízení vývojových projektů, ale i kontext a

okolnosti použití. Podrobné poznámky z rozhovorů a pozorování jsou umístěny v textové podobě na přibaleném CD. Elektronická forma na CD je agregací a přepisem ručně psaných poznámek na papír během faktického zkoumání. Výhody v zápisu na papír vidím dvě. Jednak je zápis obecně rychlejší a jednak nevyvádí respondenty tolik z míry jako slyšitelné psaní na klávesnici, natož diktafon.

U řady společností jsem zjistil, že používají odlišný přístup podle typu projektu nebo klienta. Za účelem jednotnosti se studie nezaměřuje na postupy při vedení interních projektů. Pod interním projektem je myšlen projekt, jehož výstup slouží pouze pro interní potřebu společnosti a nemá přímý dopad na zákazníka. Příkladem může být implementace vlastní vývojové platformy nebo vnitropodnikového informačního systému. V tomto případě je vývoj specifický, neb je plně v režii společnosti. Nadto, interní projekty nejsou vázány striktními termíny ze strany zákazníka a práce na nich probíhá zpravidla podle potřeby.

Rozdílnost v přístupu jsem vypozoroval i u zakázkových projektů v rámci jedné společnosti. Zvolená metodika se často liší podle schopností a možností zákazníka. Nejzásadnější proměnnou je schopnost zákazníka definovat požadavky. Mezi další faktory se řadí soulad metodiky s interními procesy zákazníka a možnost poskytnuté součinnosti. V tomto případě se ve studii omezuji na nejčastější scénář průběhu projektu v dané společnosti.

2.5.1 Softwarové inženýrství I

Bakalářský kurz Softwarové inženýrství I (BI-SI1) je povinný předmět napříč obory na FIT ČVUT. Většina studentů absolvuje předmět ve svém druhém ročníku studia. Popis v kapitole staví na informacích ze studijního portálu EDUX [23] a mé vlastní zkušenosti.

Anotace předmětu zní: *„Studenti se seznámí s metodami analýzy a návrhu rozsáhlejších softwarových celků, které jsou typicky navrhovány a realizovány v týmech. Své znalosti si upevní a prakticky ověří při analýze a návrhu rozsáhlejšího softwarového systému, který bude vyvíjen v souběžném předmětu BI-SP1*[1]*. Studenti se seznámí s CASE nástroji využívající vizuálního jazyka UML pro modelování a řešení softwarových problémů. Studenti se seznámí s problematikou objektově orientované analýzy, návrhu, architektury, metod validace, verifikace a testování.“* [24]

Pro většinu studentů představuje kurz první konfrontaci s týmovou spoluprací. Průběh projektu a jeho zadání se liší podle okolností absolvování kurzu:

1. Student má zapsáno pouze povinné BI-SI1 a studuje jeden semestr. Povinné minimum pro všechny studenty.

[1] BI-SP1 (Softwarový týmový projekt 1) a BI-SP2 jsou oborové kurzy navyšující alokaci výuky BI-SI1 formou přidělení vedoucího (učitele) a nárokem na týdenní konzultace.

2. Student má zapsáno BI-SI1 + BI-SP1 jeden semestr a následující semestr BI-SP2. Tato varianta je povinná pro studenty oboru Softwarové inženýrství.

V rámci studie se zaměřím na první variantu, protože dle studijního plánu absolvuje každý povinně BI-SI1, a tedy všichni studenti opouštějí bakalářské studium s těmito znalostmi. Jde i o variantu, kterou jsem absolvoval já.

Přednášky se zaměřují hlavně na analytickou a návrhovou část projektu. Studenti zjistí souvislosti mezi businessem zákazníka a dodávaným softwarem. Kurz se v těchto ohledech opírá o modelovací jazyk UML. V pozdějších týdnech jsou přednášky věnovány implementaci, testování a teorii metodik softwarového vývoje.

Průběh semestrálního projektu odpovídá metodice Unified Process, které jsem věnoval samostatnou kapitolu 1.2.2.1. Samozřejmě metodika je zjednodušena, aby odpovídala školnímu prostředí. Semestr je rozdělen na tři iterace oddělené milníky, kde tým odevzdává výstupy:

1. iterace. Převažuje analýza, zahájení práce na návrhu.
   - analýza business procesů *(Activity diagram)*
   - analýza požadavků *(Requirements, Use Case diagram)*
   - doménový model tříd *(Class, State Machine diagram)*
2. iterace. Převažuje návrh, doplnění analýzy, zahájení implementace.
   - návrh logické architektury *(Package, Class diagram)*
   - realizace případů užití *(Sequence diagram)*
   - databázový model *(Class diagram)*
   - implementace prototypu
3. iterace. Převažuje implementace, dokončení návrhu.
   - instalační balíček
   - instalační příručka *(Deployment diagram)*
   - vygenerovaná dokumentace ze zdrojových kódů
   - zdrojové kódy aplikace

Ukazuje se, že popsané iterace korespondují s fázovým rozdělením UP.

2.5.2 Softwarové inženýrství II

Na BI-SI1 volně navazuje kurz Softwarové inženýrství II (BI-SI2). Předmět je na rozdíl od BI-SI1 volitelný, obsahuje pouze přednášky a je vyučován pod záštitou společnosti Profinit, jednoho z lídrů zakázkového vývoje enterprise aplikací v České republice. Kapitola čerpá z vlastní zkušenosti a materiálů na stránkách Profinitu [25].

Budu-li citovat stránky Profinitu: *„Tento předmět si klade za cíl jasně a srozumitelně diskutovat základní aspekty Softwarového inženýrství v praxi, na reálných projektech, ve skutečném životě.“*

„Naší hlavní snahou je ilustrovat běžný projektový život tak, jak jej poslu-chač předmětu s největší pravděpodobností v blízké budoucnosti zažije a bude po zbytek své praxe prožívat. V průběhu semestru budou probrány jednotlivé oblasti Softwarového inženýrství (od analýzy, architektury, přes konstrukci, testování, dokumentaci, PM, atd.) vždy s potřebným teoretickým úvodem, ale současně s velkým důrazem na praxi, praktické zkušenosti a s řadou ukázek a příkladů.“ [25]

Z vlastní zkušenosti absolvování mohu potvrdit citovaný úryvek. Skutečnost má své konsekvence. Prvně BI-SI2 nerepresentuje samo o sobě metodiku, jako BI-SI1. Během první přednášky probíhá seznámení s přístupy vývoje softwaru s obdobným rozdělením, jako v kapitole 1.2. Diskutují se rozdíly metodik a vhodnost použití dle typů projektů. Filosofie předmětu se dá shrnout slovy garanta předmětu a spolumajitele Profinitu, pana Tomáše Krátkého: *„Chci, abyste o problematice přemýšleli, to mi úplně stačí.“* [26]

Zbylé přednášky jsou vedeny ve stejné filosofii a dopodrobna rozebírají jednotlivé aktivity softwarového inženýrství dle rozdělení zmíněném v kapitole 1.1 o UP. Kurz ukazuje možnosti, jak se v praxi přistupuje k softwarovému inženýrství a jaké jsou důsledky daných rozhodnutí. Z tohoto důvodu předmět nemůže vystupovat ve výzkumu jako samostatná metodika. Co ovšem může, je korigovat nabyté znalosti z BI-SI1.

BI-SI2 namítá, že v praxi je během analýzy a návrhu zbytečné vytvářet některé diagramy:

- *Use Case diagram.* Redundantní vzhledem k zadávací specifikaci a wireframes. Užitečný pouze v případě, kdy je use case roven testovacímu scénáři při akceptačním testování. Dokumentace má přinést abstraktní pohled a je doplněna samodokumentujícím se kódem.

- *Sequence diagram.* Moc konkrétní a křehký vůči změnám implementace. Lepší volbou jsou abstrahované diagramy aktivit.

- *Class diagram.* Kompletní návrhový model tříd je zbytečné vytvářet a náročný na udržení konzistence s realitou. Třídy existují ve skutečné podobě v repositáři a jsou dostupné přes IDE. V praxi se modelování

omezuje pouze na definici vnějšího rozhraní a zachycení databázového modelu.

Během BI-SI1 je testování věnována pouze jedna přednáška a během projektu se testování neřeší. BI-SI2 učí, že během analázy je vhodné vytvořit testovací plán. Dokument stanovující co, jak a kdo bude testovat. Podle kurzu se vyplatí psát unit a integrační testy. Ohledně UI má smysl automatizovat zejména smoke[2] a regresní[3] testy, protože se jedná o ideální poměr mezi množstvím odhalených chyb a náklady na automatizaci. Výhoda těchto testů je brzké odhalení chyb a jejich úspěch zaručuje základní funkčnost hlavních (často nejvíce používaných) částí systému.

Z hlediska nasazení je doporučeno mít developement, testovací a preprodukční prostředí s napojením na CI nástroj. Development prostředí slouží jako první testovací prostředí většinou spojené s automatickým buildem. Oproti lokálnímu stroji vývojáře disponuje vyšším výkonem. Testovací prostředí je oproti development plně integrováno s okolními systémy a obsahuje stabilní verzi určenou k testování nové funkcionality. Release candidate putuje na preprodukční prostředí, kde z pravidla probíhá detailní testování s využitím zamaskovaných reálných dat.

Velký důraz klade předmět na konfigurační řízení. Výstupem série přednášek je, že všechny činnosti spojené s úkoly je potřeba evidovat v issue trackeru. Cílem je být schopen zjistit kompletní stav projektu z jednoho místa. Ohledně verzování musí existovat plán stanovující co a jak se bude verzovat. Apeluje se nejen na verzování zdrojového kódu, ale i na dokumentaci, testovací data, databázové scripty nebo konfigurační soubory.

Upravená osnova projektu z BI-SI1 na základě doplnění o poznatky z BI-SI2 vypadá následovně:

1. iterace. Převažuje analýza, zahájení práce na návrhu.
    - zadávací specifikace a wireframes
    - analýza business procesů *(Activity diagram)*
    - stavové diagramy entit *(State Machine diagram)*
    - testovací plán
2. iterace. Převažuje návrh, doplnění analýzy, zahájení implementace.
    - návrh logické architektury *(Package, Class diagram)*
    - databázový model *(Class diagram)*

[2]Smoke testy povrchově testují základní funkcionalitu. U webových aplikací se jedná například o registraci, přihlášení nebo proklikání menu.

[3]Regresní testy kontrolují, zda implementace nové funkcionality nezapříčinila nefunkčnost stávajícího systému. Nutnost regresního testování je jeden z negativních atributů inkrementálního vývoje.

- implementace prototypu

3. iterace. Převažuje implementace, dokončení návrhu.
   - instalační balíček
   - instalační příručka *(Deployment diagram)*
   - vygenerovaná dokumentace ze zdrojových kódů
   - zdrojové kódy aplikace

2.5.3 Senman

Kategorie	Typ vývoje	Přístup
malá	zakázkový	smíšený

Kapitola reflektuje výstup z rozhovoru s majitelem společnosti, Davidem Černým [27], a uskutečněným pozorováním, v rámci kterého jsem mohl vidět činnost vývoje a testování [28].

2.5.3.1 Kontext

Senman se specializuje na informační a zabezpečovací systémy. Vyvíjí zejména desktopové aplikace na .NET platformě. U některých projektů dodává mimo softwarového řešení i příslušný hardware. Projekty trvají z pravidla pár měsíců. Kromě stabilních šesti zaměstnanců na projektech nárazově spolupracuje řada externistů.

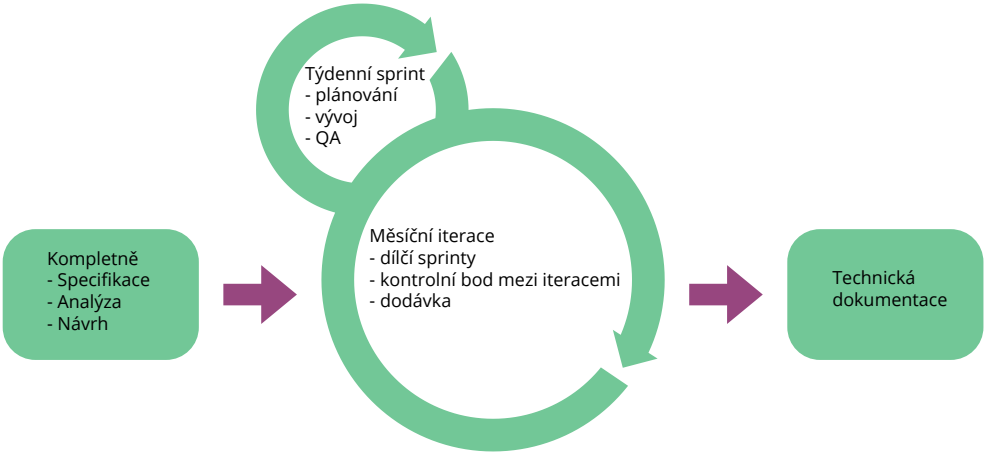
2.5.3.2 Metodika

Řízení projektů si bere z každého přístupu něco. Začátek probíhá vodopádově a zahrnuje specifikaci požadavků, jejich analýzu a následný návrh řešení. Dokumenty se tvoří sdíleně online, mimo textu se hojně používají jednoduché obrázky a fotí se nákresy z firemní tabule. Je snahou rozdělit zadání na dílčí moduly, potažmo podprojekty, které se dají přiřadit jednotlivým externistům a členům týmu. Po rozpadu zadání na jednotlivé úkoly probíhá vývoj iterativně.

Před začátkem vývoje se projektový tým setká na kick-off meetingu, během kterého se poznají i nasmlouvaní externisté. Iterace jsou rozděleny kontrolními body, jejichž součástí je předání funkčního systému. Každá další iterace znamená novou, obohacenou verzi. V rámci iterace se práce plánuje po týdenních sprintech. Iterace je dlouhá jeden měsíc, a tudíž obsahuje čtyři sprinty. Náplň sprintu určuje PM. Plánování ulehčují souvislosti mezi požadavky zpozorované při analýze. Každý týden se schází account manažer (AM) s klientem osobně, jinak komunikují denně elektronickou formou. DSM kvůli velkému počtu externistů neprobíhají.

QA probíhá den před koncem sprintu formou ručního testování. Testeři jsou lidé mimo IT prostředí, což lépe simuluje uživatelské chování. Po příchodu do práce mají připravené prostředí s testovanou verzí systému. Testované scénáře pokrývají základní a novou funkcionalitu. Větší přetestování probíhá podle potřeby. Přes interní systém testují popsané scénáře a nahlášené chyby se automaticky konvertují na bug task. Menší problémy opravují vývojáři ihned a přenasazují novou verzi.

Po ukončení implementace se tvoří technická dokumentace popisující vyvinuté řešení z technického pohledu. Mimo to se povinně dokumentuje kód a tuto skutečnost kontroluje i CI při automatickém buildu. Unit a integrační testy se píší sporadicky. Důvodem je často se měnící se kód vyžadující údržbu testů a realita, že většinu chyb odhalí ruční testování.



Obrázek 2.1: Metodika Senman

Důvody použité metodiky pro Senman jsou opatrnost, dodržení termínů a přizpůsobivost zákazníkům. Kvůli detailní přípravě na začátku a prodlevách v komunikace se někdy stává, že na implementaci zbývá méně času, než bylo původně plánováno, a vývoj začíná před ukončením analýzy.

2.5.3.3 Tým

AM, PM, UI, Vývojáři, QA

2.5.4 Ackee

Kategorie	Typ vývoje	Přístup
malá	zakázkový	agilní

Zpracovaný výstup vychází z rozhovoru se spoluzakladatelem Ackee, Dominikem Veselým [29]. Vedení Ackee následně odmítlo moji účast na pozorování.

2.5.4.1 Kontext

Ackee je mladá firma zabývající se vývojem serverových a mobilních aplikací za použití nejnovějších technologickým trendů. Spektrum klientů začíná start upy a končí nadnárodními společnostmi. Ackee si zakládá na firemní kultuře a usiluje o zlepšování chodu firmy při zachování velikosti.

Tato společnost je jedna z firem, u které neproběhlo pozorování. Samotného mě to překvapilo, vzhledem k tomu, jak se Ackee navenek presentuje. Zakladatelé společnosti jsou aktivní na FIT ČVUT a v médiích, pořádají konference a vedou volitelný předmět o projektovém řízení, který jsem mimojiné absolvoval.

2.5.4.2 Metodika

Podle slov Dominika Veselého se *„Ackee snaží vyvíjet agilně, nikoli však čistým Scrumem. Jedná se spíše o sprintové vodopády.“* [29] Projekt začíná prvotní analýzou, kde se specifikují rámcově funkční požadavky, architektura řešení spolu s výběrem technologií a dělají se wireframy. Vytváří se časové odhady a plán práce. Na analýze a návrhu se podílí dva až tři lidé, včetně člověka, který zaujímá posléze externí dohled nad projektem. Tím člověkem je CTO nebo jiný seniorní vývojář.

Vývoj probíhá po dvoutýdenních sprintech. Během sprintu se dospecifikovávají úkoly. PM komunikuje se zákazníkem elektronicky a osobně podle potřeby. Vývojáři jsou od komunikace s klientem odstíněni.

Jakmile je úkol vyvinut a projde úspěšně buildem, přechází do QA oddělení. Ackee má společné QA pro celou firmu. Testeři testují průběžně úkoly ručně po větších balících. Zákazník i QA mají pro otestování aplikaci k dispozici na testovacím prostředí. Automatizované testy se nepíší z důvodu nízké kvality nástrojů, pracnosti a časovým nárokům. Unit testy se nepíší, jsou *„dobrovolně doporučené“*. [29] Dokumentuje se zejména architektura a API. Slovy Ackee je vyvinutý software intuitivní a nepotřebuje uživatelskou dokumentaci.

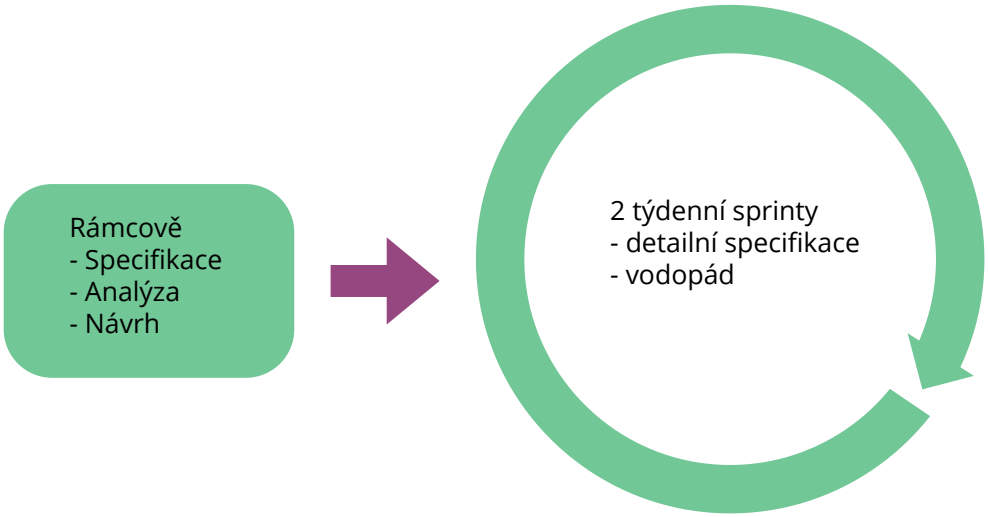
Podle Ackee přináší metodika rychlé výsledky a je zábavná pro lidi participující na projektu. Metodika je podpořena dobrou infrastrukturou (CI, Docker...) a přesahem schopností jednotlivých lidí. Ackee přiznává že průběh je méně přehledný a prostor ke zlepšení vidí v QA a dokumentování.

2.5.4.3 Tým

PM, UI, Vývojáři, Externí dohled

2.5.5 Riganti

Popsaná metodika odráží poznatky z rozhovoru se zakladatelem Riganti, Tomášem Hercegem [30] a účastněného pozorování [31]. Během dne v Riganti



Obrázek 2.2: Metodika Ackee

Kategorie	Typ vývoje	Přístup
malá	zakázkový	smíšený

jsem měl možnost interagovat s lidmi z několika projektů.

2.5.5.1 Kontext

Zaměřením Riganti je vývoj pro Windows platformu. Mimo desktopových a serverových aplikacích vytváří i webové stránky a mobilní aplikace. Klientela je ze značné části britská a zámořská, čemuž je uzpůsobena i pracovní doba, která reálně začíná po obědě a končí i v nočních hodinách. Kromě zakázek vyvíjí vlastní framework, ze kterého chce ve střednědobém plánu vytvořit produkt v podobě vývojové platformy. Projekty Riganty jsou dlouhodobé a často vývoj trvá kontinuelně i roky.

2.5.5.2 Metodika

Ukázalo se, že Riganti vyvíjí hodně ad hoc a je obtížné representovat společnost jednou metodikou. Jeden z častějších způsobů probíhá následovně. Během počáteční fáze analýzy si strany vyjasní představu a rámcové zadání. Riganti si provádí analýzu zákazníka, kde zkoumá jeho finančí zdraví. Po ustanovení rámcové představy pokračuje vývoj ve třítýdenních cyklech.

V rámci cyklu probíhá specifikace, implementace a feedback. Specifikace slouží ke konkretizaci rámcového zadání, vytvoření odhadů a plánu implementace. Hodně se využívají wireframy tvořené UI designerem. U každého projektu je ze strany zákazníka vyčleněný PO, který je týmu plně k dispozici

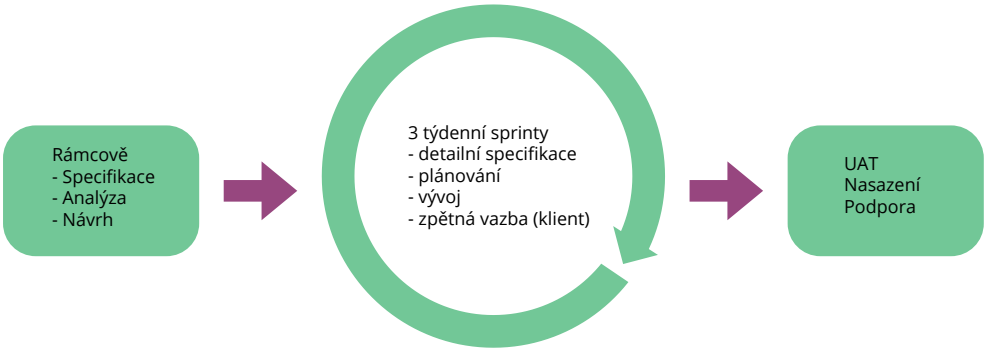
a má odpovědnost za zadávání požadavků, jejich prioritizaci a odpovídání na dotazy vývojovému týmu.

Úkoly se zakládají do product backlogu, kde přílohou je zápis ze schůze nebo korespondující emailová komunikace. Na začátku implementační fáze cyklu vytváří team leader (TL) návrh řešení a rozdělí práci mezi ostatní vývojáře. Hotové úkoly kontroluje jak z hlediska kódu (codereview), tak z hlediska funkčnosti po větších celcích na testovacím prostředí.

Tester v týmu píše automatizované testy. Automatizují se smoke testy a pouští se jako součást automatického buildu na testovací prostředí každým commitem. Nadto denně probíhá noční build. Riganti používá TFS, což je kombinace verzovacího systému, CI nástroje a Issue trackeru od společnosti Microsoft. Po ukončení třítýdenní fáze se nasazuje na stage prostředí určené pro klienta a jeho testování. Do produkce se kód dostává manuálně podle dohody. Riganti podle okolností praktikuje A/B testování.

Komunikace se zákazníkem probíhá denně elektronicky mailem nebo přes Skype se zmíněným PO. Osobní schůzky se často nekonají, protože nejvýznamnější klienti Riganti jsou ze zahraničí.

Z pohledu dokumentace se vytváří technická dokumentace popisující použité technologie, architekturu řešení a model nasazení. Uživatelská dokumentace se netvoří, protože ji zákazníci nechtějí.



Obrázek 2.3: Metodika Riganti

Riganti se nachází ve stavu učení. Společnost má kolektiv mladých lidí s věkovým průměrem 25 let. Společným rysem pro většinu lidí je nevystudovaná vysoká škola, protože dali přednost práci a brzké kariéře. Zmíněný stav učení reflektuje proces doplnění znalostních mezer i z vysokoškolského studia. Z povahy zakázkového vývoje je motivací použití metodiky spokojenost zákazníka a míra adaptability.

V čem Riganti předčilo ostatní, je vysoký developerský komfort. Drtivá většina zaměstnanců má tři širokoúhlé monitory (s méně než dvěmi jsem se nesetkal) a pohodlné herní křeslo. Asi polovina lidí vyvíjí na výkonném stolním počítači, ke kterému se mimo kancelář připojují vzdáleně, namísto v dnešní

době standardním notebooku.

2.5.5.3 Tým

TL, UI, Vývojáři, QA

2.5.6 Etnetera

Kategorie	Typ vývoje	Přístup
střední	zakázkový	vodopád

Kapitola představuje metodu projektového řízení v Etnetera na základě rozhovoru s team leaderem, Janem Randou, a projektovým manažerem Vladimírem Beranem [32]. Výstup je obohacen o poznatky z absolvovaného pozorování [33]. Během pozorování jsem byl součástí projektového týmu Marka Stefanidise. Účastnil jsem se mimojiné DSM, schůzky s klientem a měl možnost individuálně komunikovat s řadou vývojářů, testerů a manažerů.

2.5.6.1 Kontext

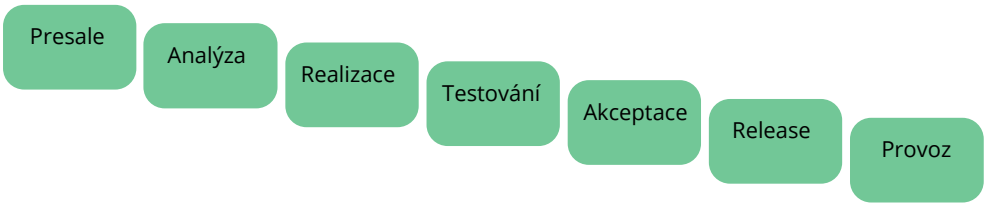
Etnetera je součástí holdingu Etnetera Group, uskupení šesti technologických firem se sídlem v pražských Holešovicích. Název společnosti vychází z latinského et cetera a jeho smyslem je trvání internetové společnosti. Zaměřením Etnetery je zakázkový vývoj webových aplikací pro přední české a slovenské korporace. Etnetera dbá na firemní kulturu, jež je nevídaná vzhledem k velikosti firmy.

Etnetera má maticovou organizační strukturu. V jedné rovině jsou lidé seskupeni do týmů. Každý tým má svého TL, jenž se stará o jejich osobní rozvoj, motivaci a pracovní spokojenost. Druhou osou jsou projekty vedeny PM, na které se alokují lidé napříč týmy podle jejich schopností a preferencí. Jednou týdně probíhá meeting, kde všichni TL a PM plánují alokace lidí.

2.5.6.2 Metodika

V minulosti usilovala Etnetera o vlastní ISO certifikaci řízení projektů. Nakonec od toho upustila a řízení projektů probíhá pomocí tzv. 14 dohod. Jedná se o soupis bodů, které musí každý projekt za svého průběhu splnit. Dohody jsou kompromisem mezi strukturou a volností. Říkají čím má projekt projít, ale je v rukou PM, jakou cestu zvolí. Zjednodušená forma 14 dohod je v příloze B.2. Nadto Etnetera vybudovala školící projektovou akademii pro své zaměstnance, kde soustřeďuje know-how z projektů.

I přes agilní trendy dnešní doby je nejčastější přístup řízení projektů vodopádový. Jeho osnova je na obrázku 2.4:



Obrázek 2.4: Metodika Etnetera

Prvotní fáze se nazývá Presale, jejím smyslem je seznámení zainteresovaných stran. Projekt je ve stavu předběžné zakázky ve výběrovém řízení. Fáze končí nabídkou s hrubou kalkulací, nástinem požadavků a možného řešení. Během této fáze je nejvíce vytížený AM, který je za nabídku zodpovědný. PM spolu s BA a technickým garantem dodávají AM podklady. Již v nabídce se diskutují požadavky na systém a tvoří se prvotní návrh technického řešení.

Analýza přichází po akceptaci nabídky a rozšiřuje určité její kapitoly (architektura, požadavky...). Analýza je rozdělena na klientské zadání a implementační analýzu. Klientské zadání obsahuje specifikace požadavků, včetně požadavků mimo scope, wireframes, případy užití, cenovou kalkulaci, odhad pracnosti a termín dodání. Pracnost odhaduje společně vývojový tým. Implementační analýza popisuje architekturu řešení, datové modely a použité technologie. Součástí analýzy je i master test plan (MTP), kde se strany domluví na průběhu QA jak z pohledu interního, tak z pohledu akceptace díla zákazníkem.

V následném období realizace vyvíjí vývojáři danou zakázku a povinně píší unit testy. Testeři píší automatizované testy a připravují se na interní testování podle MTP. Práce BA a technického garanta v omezené míře pokračuje. BA chápe doménu zákazníka a je prvním člověkem, na kterého se vývojový tým obrací s dotazy. Technický garant figuruje jako konzultant a dělá codereview technického řešení, případně se podílí jako hlavní vývojář. Poslední týmovou rolí je člověk z Deployment oddělení. Ten se stará o infrastrukturu projektu a vývojová prostředí.

Jak plyne ze 14 dohod, průběh realizace se liší tým od týmu. Některé týmy se řídí dominantně Ganttovým diagramem, jiné plně operují přes Issue tracking systém. Společnými prvky nicméně jsou pořádání DSM, využívání boards a operativní komunikace.

Interní testování dle scénářů z MTP probíhá v samostatné fázi po dokončení realizace. Obecně probíhá testování jak ruční, tak automatizované. Dalšími běžnými praktikami jsou zátěžové a penetrační testování, end-to-end a integrační testy. QA píše testy do vlastního repositáře a spolu s Deployment oddělením se stará o konfiguraci CI. Projekty mají z pravidla development a testovací prostředí. Na development prostředí se nasazuje každým commitem a je přístupné pouze interně. Naopak testovací prostředí je určeno QA a zákazníkovi, kde frekvence nasazení je minimálně jednou denně.

Po ukončení interního testování je zakázka předána k akceptaci klientovi. Po otestování klientem přichází akceptace řešení a předání projektu do provozního stavu. V provozu se o projekt stará First Line pracovník, který se zapojuje do projektu už v předešlých fázích kvůli rychlejší adaptabilitě. Výstupní dokumentace projektu se nazývá Systémová specifikace, což je spojení Implementační analýzy spolu se změnami během dalších etap projektu. Etnetera vyžaduje, aby projekty po nasazení do produkce pokračovaly fází podpory. Mezi další tvořenou dokumentaci patří popis API, provozní manuál pro FL, administrátorská příručka a v neposlední řadě uživatelská dokumentace či školení uživatelů systému.

Etnetera si zakládá na firemní kultuře, a to reflektuje i popsaná metodika. 14 dohod dovoluje projektům pružnost a nabízí otevřenost novým nápadům. Etnetera balancuje na hraně, co definovat vnitropodnikově a čemu nechat volný průběh. Jedním z úskalí flexibilnějšího přístupu se totiž ukázala být roztříštěnost technologií a používaných systémů napříč firmou. Etnetera také přiznala, že jsou více technici, než businessmani.

2.5.6.3 Tým

AM, PM, BA, Technický garant, Vývojáři, QA, Deployment, FL

2.5.7 2N

Kategorie	Typ vývoje	Přístup
střední	produktový	smíšený

Popsaná metodika vychází z informací nabytých během rozhovoru s projektovým manažerem Pavlem Rybou [6]. Pozorování se nepodařilo uskutečnit kvůli odmítnutí ze strany vedení 2N.

2.5.7.1 Kontext

Jméno 2N vzniklo z názvu ulice Na Nivách, kde společnost dříve sídlila [34]. Po více než 25 letech od založení firmy se dnes pyšní titulem jedničky na trhu v oblasti IP intercomů. Do budoucna chce 2N ještě více expandovat. Aktuálně má pobočky nejen v České republice, ale i Itálii a USA.

Hlavními projekty 2N jsou jejich produkty, klientem je trh a analýza jeho požadavků. Společnost reflektuje potřeby trhu a díry na trhu vyplňuje inovacemi. S novinkou nemusí přijít klíčoví odběratelé, ale i někdo z firmy. Zajímavostí je, že v produktech 2N existuje systém posílající o sobě zpětnou statistiku. Report je vyhodnocován a bývá 2N impulsem pro vylepšení produktu.

2.5.7.2 Metodika

Během posledních 10 let se, podle vlastních slov, 2N snaží vyvíjet agilně dle principů metodiky Scrum. Reálně se jedná o hybrid mezi iterativním a agilním vývojem. Uvolnění verze probíhá kvartálně. V tomto časovém okně se první dva měsíce vyvíjí po třítýdenních sprintech a zbylý měsíc se testuje.

Za sběr a definici požadavků je zodpovědný PO. Jejich pojmenování a definice probíhá na začátku projektu a v jeho průběhu projektu se nejvýše korigují. Nové požadavky jsou podrobeny důkladnému schvalovacímu procesu jehož součástí je analýza dopadu na projekt z finančního a časového hlediska. Softwarový architekt připravuje architektonický návrh řešení a podílí se i na realizaci v etapě vývoje.

Průběh vývoje ctí pravidla Scrumu. PO vytváří úkoly do Product backlogu, ze kterého se přelévají každý týden do Sprint backlogu na pravidelné schůzce PO a PM. U větších projektů se pořádají DSM a týdně probíhá retrospektiva týmu. Po každém release se kontroluje stav projektu a dodržení plánu se zástupci oddělení a vedením firmy.

Vývojáři dostávají úkoly od PM z týdenního sprint logu. Denně probíhá noční build nového kódu. Volba vývojových prostředí záleží na interních pravidlech týmu. Po vyvinutí úkolu probíhá povinné codereview. Následuje testováním QA oddělením, nicméně finální uzavření úkolu dělá PM.

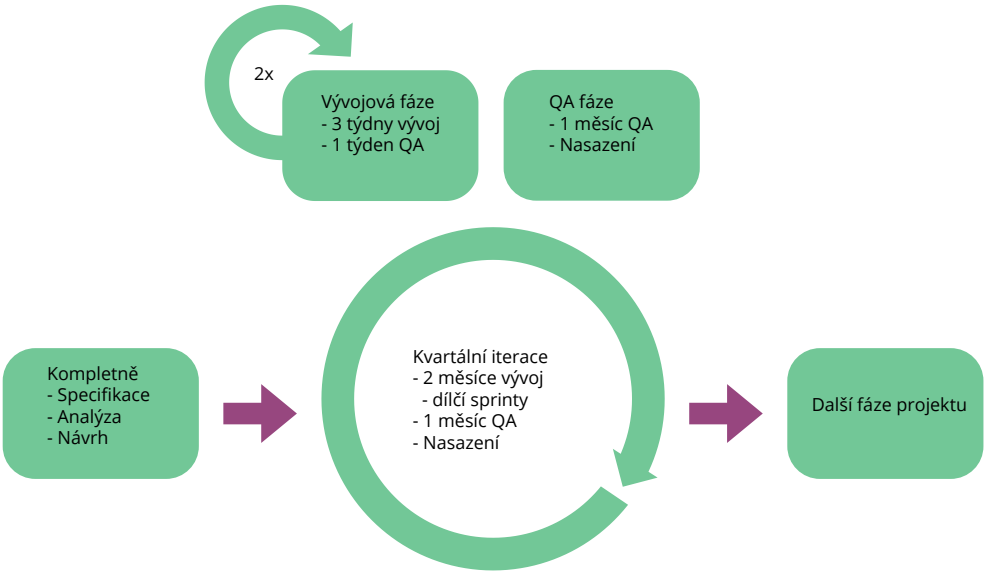
QA testuje úkoly průběžně podle tzv. developer specification. Jedná se o popis řešení úkolu, který musí každý vývojář povinně vyplnit před jeho dokončením. Ačkoli je QA oddělení společné pro celé 2N, jednotliví testeři jsou dedikování na určité projekty. Mimo testování sepisují testovací scénáře, podle kterých se testuje v období před nasazením. Kromě ručního testování píší automatizované a integrační testy.

Z povahy produktového vývoje vytváří 2N uživatelskou dokumentaci ve formě manuálů, FAQ a popis certifikátů. Manuály píše QA v souladu s testovacími scénáři. Povinně se dokumentuje API. Veškerá dokumentace je přístupná přes wiki a její správnost kontroluje PM.

V čem 2N vyniká je využití Issue trackeru JIRA. Společnost využívá vysoké konfigurovatelnosti a podporuje nastavenými flow firemní procesy. Podle slov projektového manažera Pavla Ryby, *„narážíme na její limity“* [6] . U projektu se například v každé etapě automaticky tvoří klíčové úkoly, které musí být splněny.

Ilustrovaná metodika na obrázku 2.5 popisuje jen softwarovou implementační část jinak většího procesu. Kompletní flow je vidět v příloze B.3 a jeho popis je mimo rozsah práce, ve které se omezuji pouze na stavy 4 a 5 pokrývající softwarové řešení produktu.

2N dospělo k popsané metodice přirozeným růstem firmy. Z živelného vývoje se stal organizovaný proces, jehož cílem je minimalizovat náklady, dodávat v čas a potřebné kvalitě. Prostor ke zlepšení vidí 2N ve sdílení informací napříč projekty. Řada projektů řeší stejný probém a řešení se duplikuje. Do



Obrázek 2.5: Metodika 2N

budoucna tomu 2N chce zamezit úpravou work-flow, v rámci kterého budou analýzu provádět společně lidé z více týmů.

2.5.7.3 Tým

Stranu zákazníka zastupuje marketingové a prodejní oddělení. Role:
PO, Marketingový PM, Senior technik
Za stranu vývoje je přítomen:
PM, TL, Vývojáři, QA

2.5.8 Adastra

Kategorie	Typ vývoje	Přístup
velká	zakázkový	iterativní

Kapitola odráží informace získané z rozhovoru s ředitelem bankovní divize, Davidem Kozelkou [35]. Během dne v Adastře jsem byl součástí týmu projektového manažera, Jiřího Turka, a managing consultanta, Jana Pacovského, kde jsem se účastnil i několika porad [36]. Byl jsem svědkem práce na dvou projektech, jedním ve fázi akceptačního testování před nasazením do produkce a dalším v období prvotního plánování.

2.5.8.1 Kontext

Adastra je velkým hráčem na poli informačního managementu. V rámci studie se zaměříme na pobočku v České republice, jež je součástí nadnárodního celku se sídlem v Kanadě. Z organizačního pohledu je rozdělena na čtyři divize, kde každá má na starosti přední klienty z odvětví jako jsou bankovnictví, telco & retail nebo pojišťovnictví. Adastra se specializuje na datové sklady, systémovou integraci a v neposlední řadě aplikační vývoj. Zajímavostí je, že vývojářský tým často sedí u zákazníka, protože klient zpravidla nechce zřizovat externí přístup do své sítě.

2.5.8.2 Metodika

Nejvítanějším modelem je iterativní přístup. Adastra umí agilní vývoj, jenže zákazník zpravidla *„nechce přijmout břemeno ve formě vyšší zodpovědnosti a nejasného výsledku“* [35]. Projekty Adastry jsou komplexní s vývojem a integrací řady systémů. Tato skutečnost nahrává iterativnímu přístupu, kde každá iterace znamená jednu aplikaci, jež postupně plní celkovou skládačku.

Adastra nazývá projektové členy konzultanty. Rozlišuje tři úrovně seniority: junior, standard a senior. Junioři vyžadují dohled, standardi jsou samostatní a senioři disponují přesahem a dokáží vést ostatní. Projektový tým je složen z kombinace lidí různé seniority. Ideálně jeden senior za každou část týmu (analytici, FE / BE vývojáři. . .) doplněn o standardy a juniory. Zmíněný senior je nejčastěji osobou TL.

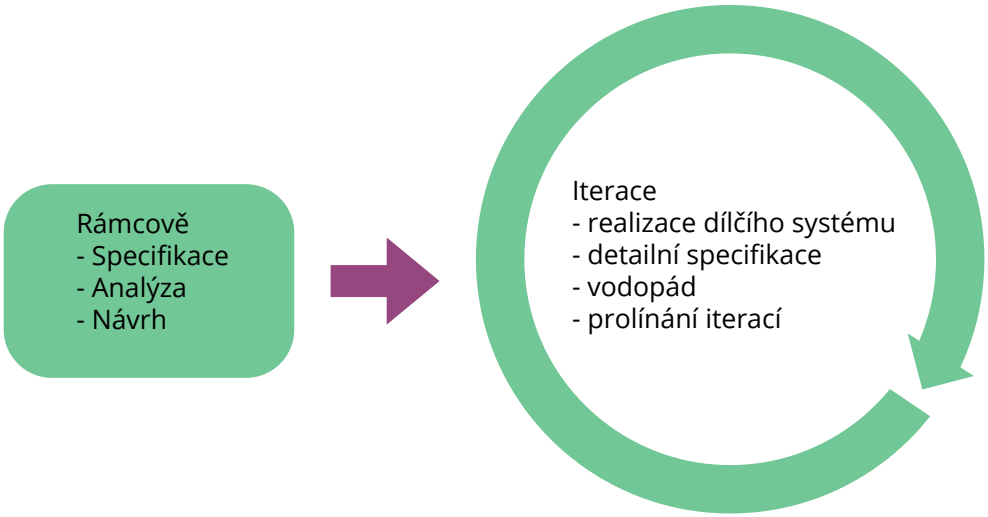
První iterace je obětována sběru požadavků a hrubé analýze řešení, jako celku. Řeší se funkčnost jednotlivých systémů a jejich vzájemná integrace. Výstupem je systémová specifikace obsahující funkční a nefunkční požadavky, rozsah projektu a akceptační kritéria spolu s výkonnostními požadavky. Na tvorbě se podílejí PM a seniorní konzultanti z řad BA i vývojářů. Senioři se po analýze a návrhu nemusí podílet na samotné fázi vývoje. Solution design je druhým dokumentem analýzy, popisující architekturu řešení. Textová podoba dokumentace je doplněna o UML modely procesů, systémovou integraci včetně definovaného rozhraní a datové flow.

Začátek každé iterace přináší hlubší specifikování a analýzu konkrétního systému. V kontraktu je zahrnuta cena časového polštáře, v rámci jehož čerpání se se zákazníkem neřeší naceňování změnových požadavků. Následují klasicky etapy vývoje, testování, akceptace a release. Průběh jednotlivých etap je předem plánován a dokumentován. Vývojáři píší povinně unit a integrační testy a dělají si codereview. Hojně se využívá mockování ještě neexistujících dílčích aplikací. QA testuje jak ručně, tak automatizovaně.

Díky převážnému využití seniorních konzultantů během analýzy se jednotlivé iterace překrývají. Při konci jedné iterace začíná souběžně analýza té následující. PM komunikuje se zákazníkem na denní bázi, týdně probíhá de-

tailní report. Analytik během realizace kontroluje postup vývoje vzhledem k UAT a podílí se povrchně na testování.

Projekty disponují typicky několika prostředími nakonfigurovanými cloudově. Jedná se o testovací prostředí s denním buildem, integrační prostředí s mocky dalších služeb a preprodukční prostředí s maskovanými daty. Release plan do produkce obsahuje kromě pozitivního scénáře i rollback plan. Mimo již zmíněné systémové specifikace a solution designu se vytváří příručka administrátora s instalačním postupem a provozní příručka s informacemi, jak udržet systém v chodu. Uživatelská dokumentace se nevytváří, protože ji zákazník nechce.



Obrázek 2.6: Metodika Adastra

Jak jsem již zmínil na začátku, hlavní motivací přístupu je ochrana spolupracujících stran. Zákazník ví co, kdy a za jakou cenu dostane. Podle jeho možností se dá parametrizovat tak, aby mu vyhovovala.

2.5.8.3 Tým

PM, TL, BA, Vývojáři, QA

2.5.9 Avast

Kategorie	Typ vývoje	Přístup
velká	produktový	agilní

Obsah kapitoly bere v potaz rozhovor s Lukášem Honzákem, ředitelem Online Operations [37]. V průběhu pozorování [38] jsem zažil práci na interním projektu i samotném produktu. Absolvoval jsem týdenní status schůzku a

plánovací meeting releasu nové verze antiviru. Mimo to jsem měl možnost hovořit se zástupci vývoje, QA i manažery týmů napříč odděleními.

2.5.9.1 Kontext

Avast je jedním z předních hráčů na poli antivirů. Výzkum se odehrával v době, kdy v Avastu probíhala integrace s konkurenčním AVG, který Avast koupil. Sloučení započalo vlnu změn a období nejistoty. Zaměřením Avastu je jejich sada produktů v čele s antivirem, označený jako In-Product. Požadavky na rozvoj jdou zejména z oddělení obchodu a marketingu. Mimo In-Product vyvíjí Avast webové stránky, pracuje na databázi, interním ERP systému nebo vlastním CMS.

2.5.9.2 Metodika

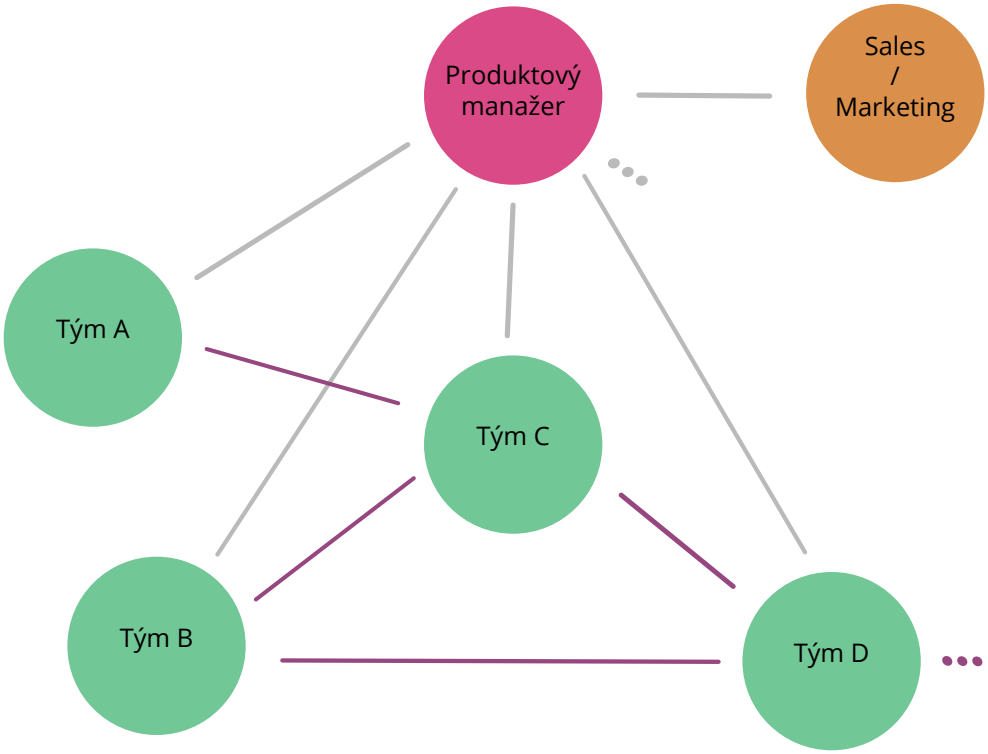
Avast vyvíjí agilně. Použitá metodika je podobná Spotify modelu [39]. Řízení probíhá produktově, to znamená, že se jedná o kontinuální vývoj předem daného rozsahu, bez pevného začátku a konce, jako tomu je u projektu. Organizace je rozdělena na autonomní týmy. Každý tým má na starost specifickou část produktu a tyto části dohromady tvoří celek. Za předpokladu, že tým dodává v čas a požadované kvalitě, je v jeho režii, jak metodicky vede vývoj [40]. Abstraktní pohled na proces jsem znázornil na obrázku 2.8.

Zpracované zadání od obchodu/marketingu se dekomponuje na dílčí úkoly pro jednotlivá oddělení. Produktový manažer (PdM) má k dispozici seznam požadavků v backlogu. Úkoly jsou seřazeny podle priority odrážející střednědobý plán vývoje. Požadavky kromě obchodu a marketingu mohou iniciovat i samotní vývojáři nebo uživatelé skrze fórum. PdM spolu s vedoucími týmů (TL) plánuje, které úkoly se dostanou do následující verze a jaký tým je realizuje. Komunikační flow znázorňuje obrázek 2.7.

Release nové verze probíhá na měsíční bázi, kde každý týden se zveřejňuje beta verze pro omezenou skupinu uživatel. Skupina se nazývá Evangelici a kromě zmíněného fóra mají možnost komunikovat přímo s vývojáři Avastu přes sdílený Slack. V případě zjištění zásadní chyby operativně vychází micro-updaty, které se aplikují uživatelům bez povšimnutí.

I přes autonomii týmů šlo při pozorování zaznamenat společné aspekty napříč týmy. Lze pozorovat, že většina týmů funguje v určité podobě Scrumu nebo Kanbanu. Metodiky jsou nicméně inspirací, nedrží se striktně všechna pravidla jmenovaných metodik. Denně probíhají stand-upy. Týmy plánují práci společně na týdenní bázi, kde úkoly v backlogu tým odhaduje a přesouvá do sprintlogu podle alokační kapacity. V rámci meetingu probíhá i retrospektiva uplynulého týdne. Hojně se využívá vizualizace progresu pomocí boardu s omezením počtu úkolů v pro daný stav.

Ostatní atributy vývoje jsou individuální a záleží na týmu, co mu vyhovuje. Setkal jsem se i s prvky feature driven developmentu, párovým programováním



Obrázek 2.7: Týmový model Avast

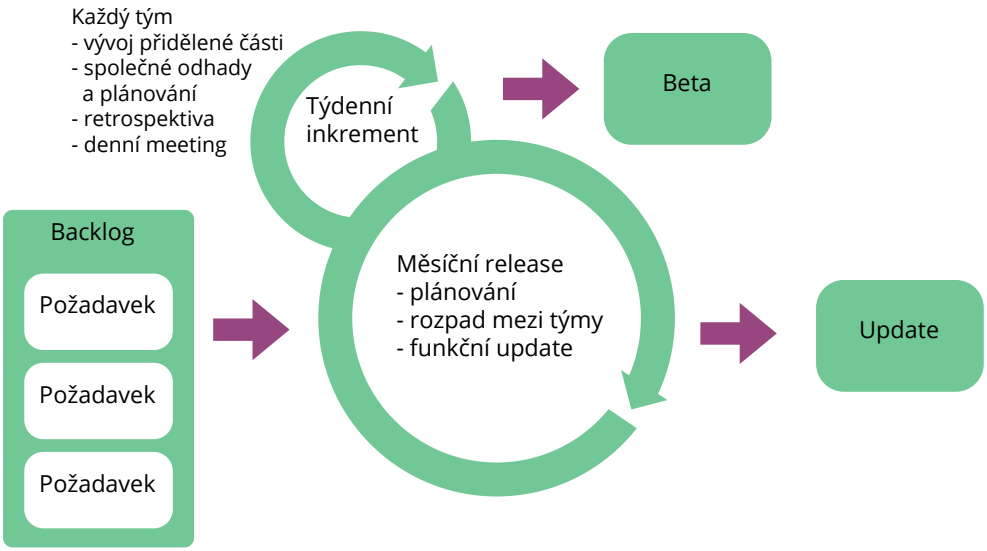
nebo relativními odhady pomocí story points. Ačkoli jsem u všech týmů viděl použití agile boardu, každý tým má vlastní stavy, pravidla přesunů i formu representace (čistě virtuální, hmotná nebo kombinace).

Týmy spolu komunikují operativně. Dokumentace k architektuře řešení se píše podle potřeby. Nejčastěji se popisuje API, případně složitější design. Tímto trendem vzniká veškerá dokumentace mimo počáteční zadání. Veškerá dokumentace je centrálně uložena ve wiki nesoucí název Centrální Mozek Lidstva (CML). V jednom systému člověk najde zadání, technickou dokumentaci, sekce jednotlivých týmů, dovolené nebo třeba firemní novinky.

Vývojáři píší povinně unit testy. QA oddělení testuje jak ručně, tak pomocí automatizovaných testů a řídí se checklistem bodů, které musí být nezbytně splněny pro release nové verze. Obsah checklistu se liší podle toho, zda se jedná o betu nebo měsíční release - pro měsíční release je obsáhlejší. QA oddělení je společné pro celý Avast, kde každý tester je dedikován na určitý produkt.

Pomocí CI se buildí před produkcí na tři prostředí. Master, stable a stage. Master představuje klasické developement prostředí s nejnovější verzí. Na stable se nasazuje denně a z povahy názvu obsahuje funkční verzi. Stage slouží jako prostředí k testování verze určené pro nejbližší release. Automatizované testy nepokrývají jenom základní průchod UI, ale i složitější scénáře, včetně

detekce infikovaného počítače podstrčenými viry.



Obrázek 2.8: Metodika Avast

Způsob vedení projektů je silně vázán na preference vrcholového managementu. Avast dříve vyvíjel projektově, měl PMO, kde byly soustředěny zdroje pro vedení a testování. Aktuálně se vyvíjí produktově. CTO má pod sebou vývojáře In-Product. CIO zastřešuje administrátory, manažery, softwarové architekty a vývojáře mimo In-Product. Existují i QA a manažeři omezení v rámci jednoho oddělení, protože dané problematice rozumí nejvíce a dokáží nejlépe vykonávat danou činnost.

2.5.9.3 Tým

PdM, TL, Vývojáři, QA
PdM a QA v rámci produktu, TL a Vývojáři tvoří autonomní týmy.

2.5.10 Shrnutí

Rekapitulace a agregace výstupu ukazuje tabulka 2.5.10. Výstupy ukazují, že většina společností volí v nějaké fázi projektu iterativní přístup. Co firmy separuje na dvě skupiny je fakt, zda specifikace spolu s analýzou probíhá před začátkem vývoje rámcově nebo kompletně. Dále se ukázalo, že společnosti nevyvíjí podle učebnicově popsaných metodik. Naopak se pojmenovanými metodikami nanejvýš inspirují a postup při řízení projektů rozvíjí podle osvědčené empirie.

Z dosavadních výstupů jsem zjistil, že menší firmy jsou více implementačně orientovány a věnují nižší pozornost disciplínám jako je analýza nebo testování. Tomu odpovídá i složení týmu, kde u větších firem se častěji objevují

Společnost	Kategorie	Typ vývoje	Přístup
Senman	malá	zakázkový	smíšený
Ackee	malá	zakázkový	agilní
Riganti	malá	zakázkový	smíšený
Etnetera	střední	zakázkový	vodopád
2N	střední	produktový	smíšený
Adastra	velká	zakázkový	iterativní
Avast	velká	produktový	agilní

separátní analytici, testeři a softwaroví architekti. Naopak u menších firem jsem se častěji setkal s kompetenčním přesahem lidí, kde projektový manažer nebo senior vývojář zastával i návrhovou, analytickou a testovací úlohu. Faktorů, proč větší společnosti kladou větší důraz i na ostatní aktivity SI jsem identifikoval několik.

1. *Procesy.* Čím je společnost větší, tím přirozeně potřebuje v nějaké formě nastavit procesy, aby chod firmy nebyl chaos. Důsledkem jsou pravidla stanovena napříč projekty, jejichž nejčastější formou je zpracování určitého dokumentu podle nastaveného postupu.

2. *Klientela.* Odběratelé, kteří jsou významní ve svém odvětví, často od dodavatele vyžadují dokumentaci, která je v souladu s jejich interními procesy. Tito klienti si často, vzhledem ke svému postavení na trhu, nemohou dovolit opomenout testování, protože fatální chyba může mít nepředvídatelné následky na business klienta.

3. *Velikost projektu.* Komplexnější projekty s velkým dopadem či zapojením více stran vyžadují více plánování a jsou náchylnější na chyby. Takové projekty jsou kvůli kapacitě zpravidla schopny realizovat větší společnosti.

Pozornost si zaslouží i rozdíl mezi produktovým a zakázkovým vývojem. Avast potvrzuje, že agilní metodiky více svědčí produktovému vývoji, kde je celý vývoj v režii samotné firmy. Odpadá role externího zákazníka v klasické podobě, byť samozřejmě zůstává zákazník ve formě impulsů z trhu, klíčových partnerů či vedení firmy. Zajímavé je, že 2N takovou agilitou nedisponuje. Skutečnost vysvětluji tím, že produkt 2N je specifický. Nejedná se o čistě softwarová řešení, jenž může mít release každý týden. 2N produkty mají hardwarovou podobu. Zmíněné flow v příloze B.3 ukazuje, že po dokončení implementace nastává proces zkušebních sérií a konstantního přehodnocování, před zahájením masivní produkce.

U zakázkového vývoje je alfou a omegou zákazník. Zákazníkovi se přizpůsobuje projektová metodika. Zájmem dodavatele většinou je, aby zákazník byl na konci spokojený, a chtěl v budoucnu uzavřít další zakázku. Jak jsem argumentoval výše, podle možností a představ zákazníka volí klient mezi levnějším

přístupem menší firmy a dražším, nicméně kultivovanějším přístupem firmy větší. Jak vystihl Jan Pacovský ze společnosti Adastra: *„Nepojedete do školky Rolls-Roycem.“* [11]

Kapitola 3

Simulace

Předmětem kapitoly je praktická část práce, ve které provádím simulaci průběhu zjištěných metodik v analytické části na ukázkovém projektu. V úvodu nejdříve vyberu metodiky vstupující do simulace. Následuje popis ukázkového projektu na kterém metodiky později simuluji. Simulace probíhá tak, že podle vybraných metodik naplánuju průběh zvoleného projektu. Výstupem jsou Ganttové diagramy, doplněné o textový popis jednotlivých úkolů, jejich vztahů, přiřazení ke zdrojům a době trvání. V závěru kapitoly je dílčí shrnutí.

3.1 Výběr metodik

Před započetím simulace je potřeba definovat metodiky, jež do ní budou vstupovat. Vzhledem k podobnosti zjištěných metodik a náročnosti provádění simulace jsem se rozhodl vybrat jednu representaci každého ze tří základních přístupů, o kterých pojednávám v kapitole 1.2. Simulace více metodik stejného typu by přineslo více námahy, než užitečných poznatků.

- *Etnetera (vodopád)* během výzkumu vykázala nejvíce prvků tradičního, vodopádového modelu. Kupříkladu metodika Senman začíná vodopádově, ale končí iterativně.

- *Kurz SI1/2 (iterativní)* jsem upřednostnil před ostatními iterativními metodikami (Adastra nebo 2N), protože se jedná právě o metodiku vyučovanou, což ji dělá unikátní v porovnání s ostatními. Výstup lze zobecnit na znalost absolventa studia FIT ČVUT. Navíc rozdíly mezi iterativními metodikami nejsou významné. Nejčastěji se jedná o délku iteračního cyklu a typ tvořené dokumentace.

- *Avast (agilní)* představují ryzí agilitu, která osvědčeně funguje v kontextu kontinuálního, produktového vývoje velké firmy. Bude zajímavé sledovat, jak se účinnost projeví na projektovém vývoji. V porovnání

s ostatními metodikami vykazující agilitu (Ackee a částečně smíšené přístupy) má Avast metodiku nejvíce propracovanou a nezanedbává ostatní disciplíny SI.

Můžete si všimnout, že jsem nezvolil metodiku, která je označená v tabulce 2.5.10 jako smíšená. Tyto metodiky jsou hůře uchopitelné a již z povahy označení vykazují aspekty více základních přístupů. Na smíšené metodiky jsem narazil častěji v menších firmách, které jsou často ve fázi procesu hledání toho, co funguje. Je potřeba dodat, že i smíšené metodiky se upínají více k jednomu přístupu, jen ne dostatečně na to, aby mohly bezpochyby nést dané označení.

3.2 Projekt

Zvolené metodiky budu simulovat na ukázkovém fiktivním projektu středního rozsahu. V IT odvětví je středním rozsahem označován projekt, jehož realizace trvá týmu čtyř lidí v řádu měsíců. Jako takový projekt volím vytvoření systému pro organizaci Logické olympiády.

Na ukázkovém projektu budou pracovat minimálně čtyři lidé: PM a tři vývojáři, kde jeden je na seniorní úrovni. Rozšíření týmu o další zdroje a role bude záviset na požadavcích použité metodiky.

3.2.1 Zadání a funkční požadavky

Systém pro organizaci Logické olympiády (SOLO) je smyšlený webový portál pro členy Mensa. SOLO zastřešuje tvorbu a evidenci testů po technické stránce. Účelem systému je sjednocení procesu tvorby nových úloh a zjednodušení sestavení testů. Funkční požadavky jsou:

1. *Tvorba úloh a podpora kolaboračního flow.* SOLO umožňuje vytvářet úlohy. Každá úloha má název, zadání, obrázek a nabídku odpovědí. Jedná se o množinu šesti odpovědí, kde jedna je správná. Na tvorbě úlohy se může podílet více lidí. Úloha může být ve stavu: Nápad, Tvorba, Grafika, Testování, Kontrola, Hotovo nebo Vyřazeno. SOLO podporuje přiřazování úlohy mezi členy a přechod mezi stavy. U každé úlohy existuje diskusní vlákno a prostor pro nahrání příloh (poznámky, náčrtky...). Úlohy, které nejsou hotové, jsou zobrazeny na nástěnce ve stylu agileboard.

2. *Kategorizace úloh podle obtížnosti a typu.* Úlohy jsou řazeny do kategorií podle obtížnosti (MS, A1, A, B a C) představující věkové kategorie řešitelů. Jedna úloha může být ve více obtížnostech, a tedy v testech pro děti různého věku. Další členění úloh je podle typu (maticová, číselná řada, geometrická...). Úloha může být označena jako rovněž testovací, tento příznak je důležitý při generování testů.

3. *Databáze a správa úloh.* SOLO obsahuje databázi úloh a nabízí nad ní operace vyhledávání, filtrování, přidávání, editaci a mazání. Hledání v databázi slouží i k zamezení tvorby duplicit. U úloh se eviduje, v jakých oficiálních testech byla použita.

4. *Správa uživatel a rolí.* SOLO není veřejně přístupný systém, uživatelé musí být platnými členy Mensa v existujícím systému MensaLide. Při přihlášení do SOLO se údaje ověřují proti systému MensaLide přes existující API. SOLO nemá separátní možnost registrace a neukládá přístupové údaje ve vlastní databázi. V rámci SOLO databáze mají uživatelé pouze přidělené role. Role jsou nezbytné pro přístup do různých částí systému a možnosti manipulace s úlohami skrze flow.

5. *Sampling úloh do testů podle kritérií.* Funkce sampling podle zadaných kritérií (počet úloh, kategorie, zastoupení jednotlivých úloh, doba od posledního výskytu v oficiálním testu...) vybírá úlohy z databáze a tvoří z nich testy. SOLO dovoluje uložit nastavení kritérií do šablon. Vytvořené testy jsou uloženy v databázi a nesou s sebou označení a datum akce, ke které patří. Testy mohou mít i příznak testovací, který slouží pro ukázku nebo použití na přípravné kurzy. Testovací testy se generují z úloh označených jako testovací.

6. *Generování testů do šablony, export a tisk.* Testy vytvořené samplingem jsou vygenerovány do grafické šablony. Do grafické šablony se kromě úloh vyplňují i identifikační údaje testu jako ročník olympiády, kolo, místo a datum konání. SOLO umožňuje výstup tisknout, exportovat ve formátu PDF nebo zobrazit ve formě náhledu.

7. *Vyhodnocení testů.* SOLO nabízí funkci pro nahrání oskenovaných, vyplněných testů a přiřazení výsledků jednotlivým účastníkům. Nad výsledky testů umožňuje SOLO provádět statistické operace a podle postupového klíče zvolit řešitele postupující do dalšího kola.

3.2.2 Změnové požadavky

V průběhu projektu vyvstanou následující změnové požadavky:

1. *Více formátů odpovědí.* 5. týden
   Původní SOLO vynucuje množinu šesti odpovědí u každé úlohy. Nově chceme mít možnost zúžit / rozšířit tuto množinu. U některých otázek potřebujeme místo množiny ponechat volné pole pro doplnění. Specifické úlohy dokonce vyžadují zákres odpovědi do nekompletního obrázku (například znázornit řez krychlí).

2. *Váhovaná obtížnost.* 6. týden
   Stávající systém dává všem úlohám například kategorie B stejnou hodnotu. Některé úlohy jsou ale těžší, než jiné, a přesto patří do stejné

kategorie. Potřebujeme zavést bodovaný systém úloh 1 až 5. Bodování se projeví do samplování testů, kde požadujeme, aby součet bodů za úlohy se rovnal celkovému počtu bodů. O bodování chceme rozšířit i kritéria samplingu, abychom mohli nastavit například pro finále, aby se nevyskytovaly úlohy obtížnosti 1 a 2 a aby úlohy obtížnosti 5 byly minimálně dvě. Úlohy spadající do více kategorií musí mít možnost nastavit obtížnost zvlášť pro každou kategorii. Úloha s obtížnosti 5 v kategorii A se může příkladem objevit v kategorii B s váhou 3.

3. *Ruční změna samplingu.* Po 1. testování funkcionality sampling
   Generování testů je pěkné, ale ne vždy nám vyhovuje. I přes nastavitelnost se nám některé testy nelíbí. Chceme mít možnost úlohy v testech přeuspořádat a ručně měnit za jiné.

4. *Změna šablony testů.* Po 1. testování funkcionality generování
   Chceme mít možnost měnit šablonu, do které se generují testy. Dokážeme šablonu připravit v HTML s definovanými prvky a tu do SOLO následně importovat. Funkce generování úloh by mi měla dát na výběr, kterou šablonu chci použít. Potřebujeme i modul pro správu těchto šablon.

5. *Více správných odpovědí.* 11. týden
   Ještě by se nám hodilo mít u úloh s výběrem z množiny odpovědí styl multiple-choice. Úloha může mít 0 až n správných odpovědí.

3.2.3 Překážky

Každý projekt neprobíhá podle představ a musí čelit nečekaným okolnostem. Během realizace nastanou tyto jevy:

1. *Pozdní nástup juniorního vývojáře.* 1. až 5. týden
   Jeden z vývojářů nastoupí do projektu až od 5. týdne po dokončení zkouškového období.

2. *Zákazník přestane komunikovat.* 8. až 9. týden
   Zákazník odjel s rodinou na 14 dní na dovolenou k moři. Nechce být rušen, nebere telefon a neodpovídá na mail.

3. *Onemocnění seniorního vývojáře.* 14. týden
   Seniorní vývojář dostal rýmu a týden může omezeně pracovat z domu v rozsahu 1 md.

Přirozeně aplikace vyvíjí lidé a ti dělají chyby. Uvažuji, že 20 % vyvinuté funkcionality obsahuje chybu. Podle typu a množství testování se liší včasnost odhalení chyby a šance opětovného výskytu.

3.2.4 Časový harmonogram

Projekt je zahájen v pondělí 1. května 2017. Strany jsou dohodnuty na spolupráci a dodavatel po prvním dni má sepsané funkční požadavky jako uvádí kapitola 3.2.1. Deadline pro dokončení projektu je pondělí 2. října 2017. Na realizaci má tedy tým pět měsíců. Dokončením se myslí release do provozu všech požadovaných částí systému.

3.3 Společné aspekty simulace

Části, které jsou společné nebo se týkají více simulačních případů, zmíním v této kapitole a budu se na ně později odkazovat.

3.3.1 Časové odhady

Nejvíce prostoru zde věnuji metodice určování odhadů. Jednotkou pro odhad je man-day (md), ekvivalent 8h pracovnímu dni. Pokud nebude psáno jinak, odhadovanou časovou náročnost stavím na základě osobní empirie.

Hlavním pramenem, ze kterého čerpám, jsou znalosti nabyté během studia kurzů bakalářského programu FIT ČVUT, jmenovitě Softwarové inženýrství I a II, Databázové systémy, Objektové modelování a Tvorba informačních systémů. Většinu těchto kurzů provází tvorba semestrální práce, jenž se zaměřuje na analytickou až implementační část projektu.

Druhým pramenem je má pracovní zkušenost, neboť v době psaní této práce jsem druhým rokem na pozici vývojáře s analytickým přesahem ve společnosti Quanti. V práci jsem mimo samotného psaní kódu konfrontován s databázovým návrhem, tvorbou dokumentace v UML a BPMN notaci, integračním testováním a příležitostné participaci na schůzkách se zákazníkem.

U položek, kde se nedokážu odkázat na jeden ze dvou výše uvedených pramenů, problém dekomponuji na co nejmenší celky a tyto dílčí celky odhaduji, potažmo si jako součást simulace vyzkouším jejich částečnou realizaci. Tyto skutečnosti mě dělají způsobilým pro tvorbu odhadů, které nicméně jsou a zůstávají pouze jenom odhady.

3.3.1.1 Specifikace

Jak jsem již zmínil u definice ukázkového projektu, vytvoření počáteční představy v rozsahu 3.2.1 trvá týmu 1 md. Tento rozsah je nedostačující pro jakoukoli hlubší analýzu nebo rozsáhlou implementaci. Následující body je potřeba dospecifikovat:

- *FP 1 + 3, Úlohy a kolaborační flow (4 + 1 md).* U úloh je potřeba dodefinovat zejména položky a jejich datové typy. Zvláštní pozornost si žádají přílohy a odpovědi. Kolaborační flow je ještě složitější. Musí se definovat přechodová funkce stavu u úlohy, počáteční a koncové stavy,

autorizační omezení. Dalším důležitým aspektem je přiřazení úloh na uživatele, vazba přiřazení na oprávnění a stav úlohy. V neposlední řadě bod zahrnuje definici nástěnky, jeji funkcionalitu, formu, možnost konfigurace a filtrování.

- *FP 2, Kategorie (0,5 md).* Tady je to přímočaré, potřebuji znát seznamy kategorií a typů úloh, plus kardinalitu na úlohu.

- *FP 4, Správa uživatel (2 md).* Zde je potřeba definovat, jak a které údaje získává SOLO z MensaLide (popis API a modelu komunikace) a jaké údaje si ukládá interně. Další otázkou je okamžik vytvoření přístupu do SOLO. Mají přístup automaticky všichni z MensaLide, musí se registrovat nebo je systém registruje sám při prvním přihlášení? Může být uživatel deaktivován v SOLO, ale stále být aktivní v MensaLide? Pro úplnost ještě potřebuji definovat uživatelské role a oprávnění k přístupu do určitých sekcí systému. S tím je spojeno i přezkoumání konzistence s kolaboračním flow v prvním bodu.

- *FP 5, Sampling (1,5 md).* Potřebuji definovat povinná a volitelná data testu (název, datum a kolo akce...). Dále seznam a datové typy kritérií pro sampling, specifikaci dat ukládaných do znovupoužitelné šablony.

- *FP 6, Generování testů (0,5 md).* Jednoduché, neznám podobu a formát šablony.

- *FP 7, Vyhodnocení testů (0,5 md).* Proces zadávání účastníků, definice postupového klíče a vyhodnocovacích statistik.

Dospecifikace požadavků v součtu zabere dalších 10 md. Výstupem je textová podoba zadávací specifikace, na jejímž základě může bez problému proběhnout detailní analýza a kompletní implementace.

3.3.1.2 Návrh

Technickým řešením je třívrstvá aplikace rozdělena na SOLO FE, SOLO BE s aplikační logikou a databázovou vrstvu.

Ruku v ruce s návrhem architektury jde výběr technologií, knihoven a frameworků. Volba programovacích jazyků se volí dle vhodnosti a znalosti vývojářů. Více času zabere vybrání pomocných knihoven a frameworků usnadňující vývoj, které vyžadují širší rešerši. Protože se jedná o standardní projekt bez speciálních nároků, pokládám 2 md za dostatečné.

V databázovém modelu jsem napočítal 15 klíčových entit: 1. úloha, 2. stav úlohy, 3. uživatel, 4. role, 5. příloha úlohy, 6. diskuse úlohy, 7. typ úlohy, 8. kategorie úlohy, 9. odpověď úlohy, 10. test, 11. realizace testu, 12. výsledek testu, 13. událost, 14. kolo události a 15. postupový klíč události. Vzhledem k náročnosti jsem si entity načrtnul pouze na papír se základními vazbami a

nemodeloval je v konceptuálním nástroji, ani se hlouběji nezabýval funkčností modelu.

U některých metodik je při návrhu potřeba tyto entity identifikovat, navrhout vzájemné vazby a pomocné tabulky. Při znalosti analýzy odhaduji vytvoření návrhu na 1 md a zanesení do konceptuálního nástroje další 1 md. Dokumentace modelu v elektronické podobě ulehčí transformaci modelu na databázové scripty.

Návrh rozhraní mezi MensaLide a SOLO BE je přímočarý. Jedná se o jeden již existující, přihlašovací zdroj, kde stačí získat dokumentaci a ověřit použitelnost. Situace mezi FE a BE částí SOLO je složitější. Vytvoření RESTful rozhraní obnáší při 15 entitách popsání 60 zdrojů. Vazba počtu zdrojů na databázový model je, že jedna databázová entita odpovídá v průměru čtyřem zdrojům (post, get, update a delete). Namapování entit a vzájemné vazby ukazuje tabulka 3.1. Zpracování jednoho zdroje zabere v průměru 30 minut. Zpracováním myslím návrh HTTP požadavku spolu, vymezení autentizace/autorizace a samotnou dokumentaci. Celková náročnost je po zaokrouhlení nahoru rovna 4 md (60 * 30 minut za SOLO FE/BE a 2 h za MensaLide).

3.3.1.3 Implementace

Pro unifikaci, a určitou vědeckou vyjádřitelnost, přepočítávám náročnost implementace na REST API zdroje. Stanovuji distribuci zdrojů podle obtížnosti na: 10 % těžké (5 md), 30 % středně těžké (3 md) a zbylých 60 % jednoduché (1 md). Jedná se o v praxi běžné rozdělení náročnosti zdrojů. Jednoduchým zdrojem jsou metody, které pouze přehazují data z jedné strany na druhou (výpis úloh na nástěnce). Středně obtížné zdroje se vyznačují parametrizací a ošetření více možných scénářů (editace úlohy). Těžké zdroje vyžadují implementaci stěžejní business logiky aplikace (přechodová funkce stavu úlohy).

Shrnuto, náročnost implementace n zdrojů odpovídá vztahu:

$$n * (0.1 * 5 + 0.3 * 3 + 0.6 * 1) = 2 * n \quad md \tag{3.1}$$

Do náročnosti je započítán i čas na otestování vývojářem. Pro zúžení komplexnosti simulace se zabývám výhradně náročnosti implementace BE části aplikace. Do času započítávám i jednoduchou implementaci FE využívajících existující šablony.

Vyjmenované entity jsem rozdělil mezi definovaných sedm funkčních požadavků, čímž dostávám jejich implementační náročnost uvedenou v tabulce 3.1. Samozřejmě některé entity jsou potřeba napříč více požadavky, což přináší logickou návaznost daných požadavků. Počet entit ve zmíněné tabulce vynásobený čtyřmi je n vstupující do definovaných rovnic pro výpočet odhadu.

Ve vymezení překážek projektu 3.2.3 uvádím, že 20 % vyvinuté funkcionality obsahuje chybu. Pro doplnění, náročnost opravy chyby stanovuji na 25 % náročnosti vývoje. Oprava chyby zahrnuje i vylepšení testování zaručující, že chyba se již znovu nevyskytne. Vyjdu-li z rovnice 3.1 pro výpočet náročnosti

Tabulka 3.1: Rozdělení hlavních entit mezi FP

Označení	Název	Entity	Závisí na
FP 1	Tvorba úloh a flow	3 (stav, příloha, diskuse)	FP 3, 4
FP 2	Kategorizace úloh	2 (typ, kategorie)	FP 3
FP 3	Databáze a správa úloh	2 (úloha, odpověď)	
FP 4	Správa uživatel a rolí	2 (uživatel, role)	
FP 5	Sampling	3 (test, kolo, událost)	FP 3
FP 6	Generování testů	1 (realizace)	FP 5
FP 7	Vyhodnocení testů	2 (klíč, výsledek)	FP 5

implementace a uvážím zmíněné faktory, náročnost opravy zdroje odpovídá vztahu:

$$(2 * n) * 0.2 * 0.25 = 0.1 * n \quad md \tag{3.2}$$

Časová náročnost vypočtená z uvedených rovnic je normalizována na jednoho seniorního vývojáře. Vývojový tým obsahuje vývojáře tři, jednoho seniorního a dva juniorní. Počítám, že dvoučlenný tým (senior + junior) bude mít výkonnost 150 % a kompletní tým 200 %. Oba junioři tedy podají stejný výkon jako samotný senior.

3.3.1.4 Testování

Aplikace SOLO je definována sedmi funkčními požadavky. V průměru počítám, že jeden požadavek se dá rozložit na pět komplexnějších testovacích scénářů. Předmětem UAT testování je tedy po vynásobení 35 scénářů.

Některé metodiky vytvářejí určitou obdobu testovací plánu. Testovací plán zahrnuje popis jak interního, tak akceptačního testování. Konkrétně vazbu testování na vývojová prostředí, harmonogram testování, testovací sadu dat, testovací scénáře a akceptační kritéria. Tvorba dokumentu po rozepsání zmíněných částí odhaduji na 3 dny.

Při vytvoření testovacího plánu předpokládám, že dodavatel zná akceptační testy. Důsledkem toho je, že pokud probíhá interní testování před UAT testováním, QA testuje interně podle akceptačních testů a po opravě chyb nahlášených během interního testování již UAT nové chyby neodhalí. QA pracovník je navíc v diagramech alokován kromě interního testování i při UAT testování na straně zákazníka, kde hraje roli jako podpora tamních testerů.

3.3.2 Výstup simulace

Primárním výstupem simulace je znázornění průběhu projektu pomocí Ganttova diagramu. Kvůli vyšší přehlednosti rozděluji diagramy podle jednotlivých etap projektu. Diagram obsahuje názvy položek, časové odhady, souvislosti úkolů a přiřazení na příslušné zdroje. V diagramech neznázorňuji práci PM.

Simulace se zabývá výhradně primárními aktivitami softwarového inženýrství a PM hraje roli podpůrnou s konstantním vytížením. Zanesení paralelních činností PM by jenom zesložitilo čitelnost diagramů, bez vlivu na simulaci, jako takovou. Výjimkou jsou situace, kdy PM má přesah do ostatních kompetenčních rolí (například BA, seniorní vývojář potažmo QA).

3.4 Etnetera

Následující simulace vychází z popisu metodiky pro řízení projektů ve společnosti Etnetera 2.5.6. Jak jsem zmínil v jejím popisu, jedná se o metodiku založenou na vodopádovém přístupu. I přes tento fakt negeneruje metodika tolik dokumentace, jak by se očekávalo. Důvodem je nejspíše empirie z praxe, při kterém společnosti redukují dokumentaci na tu, kterou opravdu potřebují.

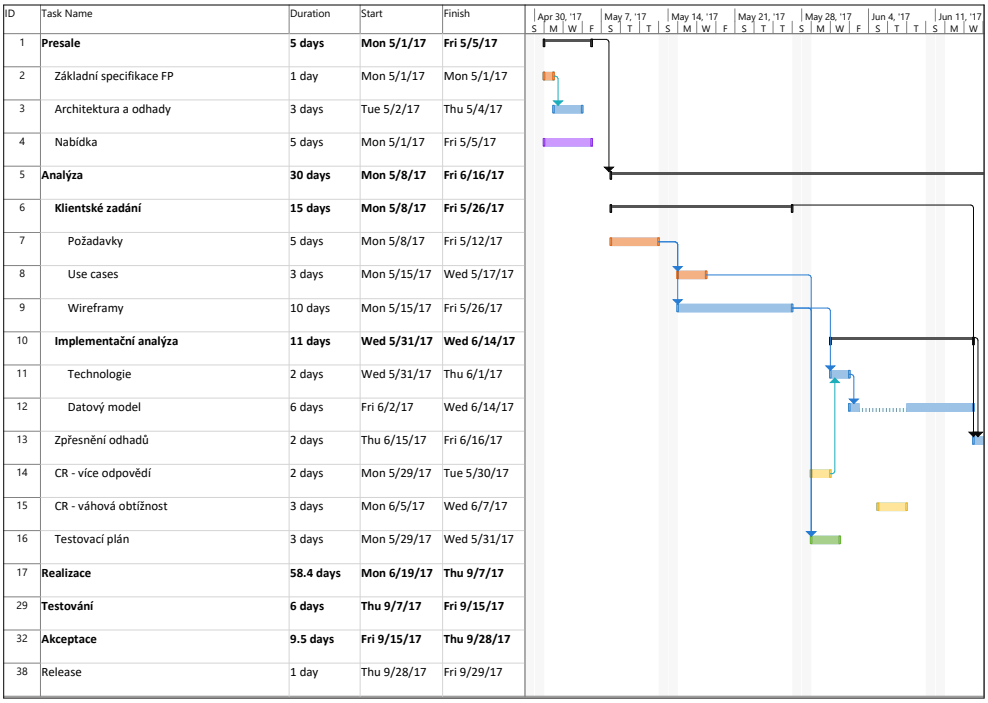
Sedmičlenný projektový tým je rozšířen o AM, BA a zástupce QA. Jedná se o zjednodušený scénář vzhledem k rozsahu projektu. V praxi Etnetera u větších projektů rozděluje analytiky na businessové a technické a oba typy angažuje zároveň. Návrh architektury zastřešuje technický garant, který později nutně nemusí hrát roli seniorního vývojáře. Protože nesimuluji sekundární činnosti SI, vypouštím v diagramech práci zástupce Deployment oddělení (infrastruktura). Vynechávám i First line pracovníka, jenž standardně přebírá projekt ve fázi provozu. Důvodem je, že projekt pro mě končí okamžikem release a částí podpory se již nezabývám.

3.4.1 Presale & analýza

V rámci presale je úkolem AM vytvořit nabídku. Jejím obsahem jsou mimo jiné počáteční odhady, harmonogram, návrh architektury a nástin požadavků. Tyto podklady dodávají obchodníkovi PM a BA. Při paralelizaci úkolů mezi tyto tři osoby odhaduji vytvoření nabídky na 5 md, protože nabídka je pouze náčrtem, který se během nadcházející fáze upřesní. Zatímco AM pracuje na nabídce, získá první den BA od klienta základní představu o funkčnosti SOLO v rozsahu 3.2.1. Na to navazuje PM, kterému senior dodá během 3 md návrh architektury a prvotní časové odhady.

Analýza rozšiřuje zmíněný obsah nabídky. Mezi její části patří klientské zadání, implementační analýza a testovací plán. Nejdříve musím začít pracovat na klientském zadání, které mi říká, co budu po dodávat. Součástí je specifikace požadavků, model případů užití a wireframe model.

Specifikace požadavků je textový popis zadání. V tuto chvíli již mám o požadavcích představu z nabídky a pouze je doplňuji. Podle 3.3.1.1 má trvat specifikace 10 md. Odhad snižuji na polovinu ze dvou důvodů. Jednak jsem část informací sesbíral během komunikace obchodníka při tvorbě nabídky a jednak je u této metodiky stěžejní částí specifikace wireframe model. Textová specifikace tedy není tak podrobná, protože část informací by byla s WF redundantní. Potom co BA ukotví zadání, může začít tvořit případy užití.



ID	Task Name	Duration	Start	Finish
1	**Presale**	**5 days**	**Mon 5/1/17**	**Fri 5/5/17**
2	Základní specifikace FP	1 day	Mon 5/1/17	Mon 5/1/17
3	Architektura a odhady	3 days	Tue 5/2/17	Thu 5/4/17
4	Nabídka	5 days	Mon 5/1/17	Fri 5/5/17
5	**Analýza**	**30 days**	**Mon 5/8/17**	**Fri 6/16/17**
6	**Klientské zadání**	**15 days**	**Mon 5/8/17**	**Fri 5/26/17**
7	Požadavky	5 days	Mon 5/8/17	Fri 5/12/17
8	Use cases	3 days	Mon 5/15/17	Wed 5/17/17
9	Wireframy	10 days	Mon 5/15/17	Fri 5/26/17
10	**Implementační analýza**	**11 days**	**Wed 5/31/17**	**Wed 6/14/17**
11	Technologie	2 days	Wed 5/31/17	Thu 6/1/17
12	Datový model	6 days	Fri 6/2/17	Wed 6/14/17
13	Zpřesnění odhadů	2 days	Thu 6/15/17	Fri 6/16/17
14	CR - více odpovědí	2 days	Mon 5/29/17	Tue 5/30/17
15	CR - váhová obtížnost	3 days	Mon 6/5/17	Wed 6/7/17
16	Testovací plán	3 days	Mon 5/29/17	Wed 5/31/17
17	**Realizace**	**58.4 days**	**Mon 6/19/17**	**Thu 9/7/17**
29	**Testování**	**6 days**	**Thu 9/7/17**	**Fri 9/15/17**
32	**Akceptace**	**9.5 days**	**Fri 9/15/17**	**Thu 9/28/17**
38	Release	1 day	Thu 9/28/17	Fri 9/29/17

Obrázek 3.1: Gantt fáze Presale a Analýza metodiky Etnetera

Modelování zahrnuje krokový a obrázkový popis scénáře v UML notaci. Když vyjdu z předpokladu, že případy užití jsou rovny testovacím scénářům při UAT testování, mám 35 scénářů. Vymodelování jednoho scénáře pomocí Use case diagramu zabere 40 minut, do času počítám krokový popis scénáře, nastavení podmínek a počátečních omezení, namapování na účastníky a vytvoření podpůrných, opakujících se scénářů. Po vynásobení a zaokrouhlení mám BA vytíženého na 3 md.

Paralelně s tvorbou případů užití pracuje seniorní vývojář na wireframech. Wireframy jsou hlavním bodem klientského zadání. Proto je vytvořený model dynamicky proklikatelný, umožňuje vyplňovat formulářová pole, zobrazuje modální okna a kontextové nabídky. U statického WF modelu jsem napočítal 20 obrazovek, při uvážení variant s kontextovou nabídkou nebo modálním oknem jsem počet vynásobil pěti. Vytvoření jednoho statického WF mi zabralo 30 minut (ukázka tří prolinkovaných WF, použitých pro odhad, jsou součástí přibaleného CD). Počáteční náročnost dynamických WF je vysoká, ale 80 % obrazovek jsou deriváty, a proto v průměru jedna obrazovka zabere 45 minut, celkově po zaokrouhlení 10 md na celý model.

Po hotovém klientském zadání mám podrobně definované požadavky, případy užití a wireframe model. QA na základě výstupu sestaví testovací plán, 3 md dle odhadu 3.3.1.4. V tento okamžik vznese zákazník první změnový požadavek (více odpovědí). Sice jsem na projektu nezačal implementovat, ale

musím úpravu zanést do požadavků, případů užití a wireframe modelu, kvůli kterému úprava zabere 2 md.

Nyní má senior příležitost začít na implementační analýze. Díky kompletnímu zadání dokáže vybrat optimálně technologie za 2 md. Tvorba datového modelu zahrnuje popis rozhraní mezi aplikacemi a model databáze, což zabere dohromady 6 md. Předešlé odhady vychází z argumentace kapitoly 3.3.1.2. Naneštěstí během tvorby modelu přichází další změnový požadavek (váhová obtížnost). Tentokrát se chyba zpropaguje i do rozpracovaného datového modelu, proto je náročnost vyšší, než předešlý změnový požadavek. Práce na datovém modelu je během požadavku přerušena a pokračuje po jeho dokončení. Posledním bodem iterace je na základě hotové analýzy zpřesnit odhady pracnosti a stanovit cenu. Aplikaci mám zdokumentovanou z různých úhlu pohledu a mohu tedy přesněji a jasněji odhadnout jednotlivé funkční požadavky během 2 md.

Presale fáze projektu trvala 5 dní a připravila základ pro detailní analýzu, která trvala 30 md. Během analýzy byl nejvíce vytížen seniorní vývojář, jenž zpracoval kompletní implementační analýzu, odbavil změnové požadavky a vypracoval wireframe model. BA vypracoval zadání spolu s use case modelem a je druhou nejvytíženější osobou týmu.

3.4.2 Realizace

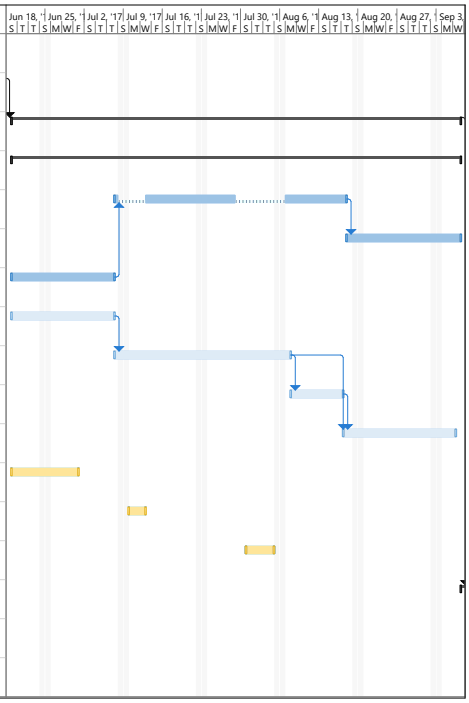
Fáze realizace představuje kompletní implementaci SOLO na základě vypracované analýzy. Jednotlivé funkční požadavky jsou poskládány podle logické návaznosti do dvou větví, kde na jedné pracuje seniorní vývojář a druhou společně vyvíjí junioři, kteří dohromady pracují stejným tempem jako seniorní kolega. Časový odhad respektuje rovnici 3.1 s rozdílem, že námaha spojená s vypracováním dynamického wireframe modelu mi ušetří 10 % času implementace. Důvodem je, že vývojáři snadněji pochopí zadání, dokáží si lépe představit požadavky a mezi-entitní vazby. Odhady tedy vychází z upravené rovnice 3.3

$$(2 * n) * 0.9 = 1.8 * n \quad md \tag{3.3}$$

Během začátku realizace odjede klient na dovolenou a není dostupný. Štěstí je, že součinnost klienta jsme potřebovali do minulého týdne, kdy tým úspěšně dokončil analýzu. Shodou okolností tedy výpadek klienta projekt neovlivní.

O dva týdny později přijde další změnový požadavek (více odpovědí II.), jenž má kromě dokumentace i vliv na započatou implementaci. Požadavek ale není tak závažný, jako požadavky během analýzy, a proto zůstávám u odhadu 3 md. Poslední komplikací během realizace je onemocnění seniorního vývojáře. Vlivem nemoci je posunuta práce na jeho vývojové linii, která v součtu trvá déle než linie juniorů. Etapa realizace se ve výsledku o den zpozdí.

ID	Task Name	Duration	Start	Finish
1	**Presale**	**5 days**	**Mon 5/1/17**	**Fri 5/5/17**
5	**Analýza**	**30 days**	**Mon 5/8/17**	**Fri 6/16/17**
17	**Realizace**	**58.4 days**	**Mon 6/19/17**	**Thu 9/7/17**
18	**Implementace**	**58.4 days**	**Mon 6/19/17**	**Thu 9/7/17**
19	FP 1 - Tvorba úloh a kolaborační flow	21.6 days	Fri 7/7/17	Thu 8/17/17
20	FP 2 - Kategorizace úloh	14.4 days	Fri 8/18/17	Thu 9/7/17
21	FP 3 - Databáze a správa úloh	14.4 days	Mon 6/19/17	Fri 7/7/17
22	FP 4 - Správa uživatel a rolí	14.4 days	Mon 6/19/17	Fri 7/7/17
23	FP 5 - Sampling	21.6 days	Fri 7/7/17	Mon 8/7/17
24	FP 6 - Generování testů	7.2 days	Tue 8/8/17	Thu 8/17/17
25	FP 7 - Vyhodnocení testů	14.4 days	Thu 8/17/17	Wed 9/6/17
26	Klient mimo	10 days	Mon 6/19/17	Fri 6/30/17
27	CR - více odpovědí II.	3 days	Mon 7/10/17	Wed 7/12/17
28	Senior nemoc	5 days	Mon 7/31/17	Fri 8/4/17
29	**Testování**	**6 days**	**Thu 9/7/17**	**Fri 9/15/17**
32	**Akceptace**	**9.5 days**	**Fri 9/15/17**	**Thu 9/28/17**
38	Release	1 day	Thu 9/28/17	Fri 9/29/17



Obrázek 3.2: Gantt fáze Realizace metodiky Etnetera

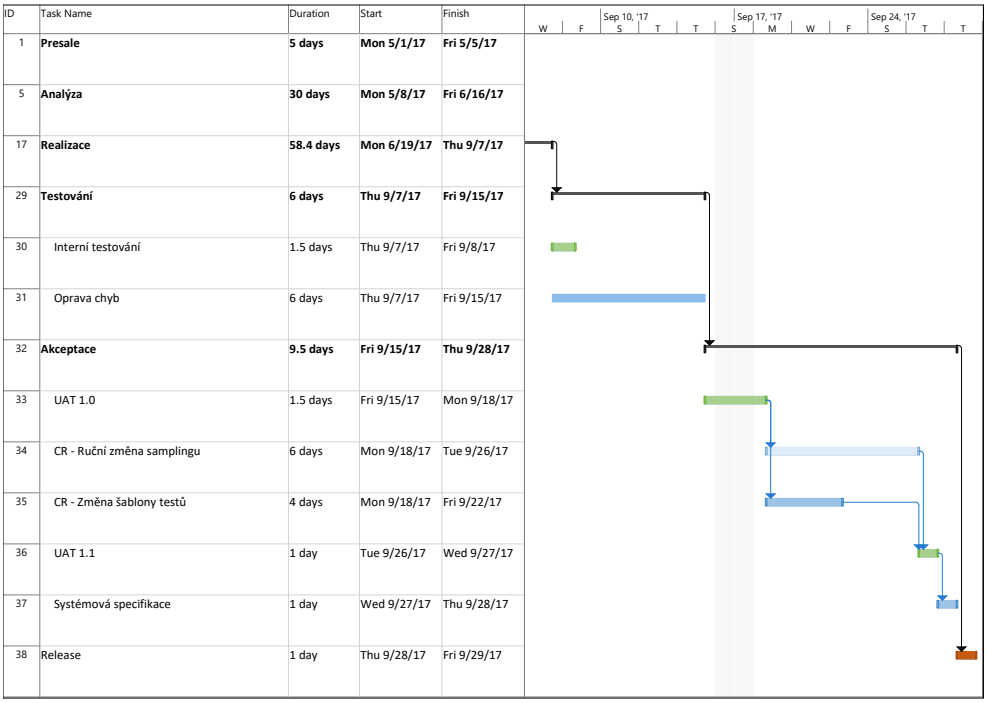
I přes dvě komplikace trvá realizace 58,4 md. Kladnou zásluhu na tom má paralelizace do dvou linií mezi seniorního a juniorní vývojáře spolu s ušetřeným časem na vývoji díky pečlivé dokumentaci.

3.4.3 Testování, akceptace & release

Etapa testování pokrývá ověření scénářů, které bude zákazník testovat při UAT. Testování 35 scénářů probíhá podle testovacího plánu a vymodelovaných případů užití, kde je přesně krok po kroku sepsáno očekávané chování systému. Vzhledem k tomu odhaduji otestování jednoho scénáře na 20 minut včetně případného reportu. Celkově se dostávám na 1,5 md.

Hotová aplikace má 120 zdrojů a dle vzorce 3.2 narazí QA na chyby v celkovém objemu práce 12 md. Díky tomu, že mám volné všechny vývojáře a testuje vyčleněný QA, jsem schopen začít s opravami operativně se začátkem testování a díky souběžnosti vývojářů se zkrátí práce z 12 md na polovinu. Interní testování tedy trvá 6 md.

Přetestování klientem ve fázi akceptace trvá rovněž 1,5 md. Po otestování klient sice nereportuje chyby, ale přijde s dalšími změnovými požadavky (ruční sampling, šablony testů). Stejně jako u opravy chyb, jsem schopen zapracování požadavků rozdělit mezi seniorního a juniorní vývojáře, a odbavit je tedy paralelně. Protože mám již vyvinutou celou aplikaci, musím úpravu kromě



ID	Task Name	Duration	Start	Finish
1	**Presale**	**5 days**	**Mon 5/1/17**	**Fri 5/5/17**
5	**Analýza**	**30 days**	**Mon 5/8/17**	**Fri 6/16/17**
17	**Realizace**	**58.4 days**	**Mon 6/19/17**	**Thu 9/7/17**
29	**Testování**	**6 days**	**Thu 9/7/17**	**Fri 9/15/17**
30	Interní testování	1.5 days	Thu 9/7/17	Fri 9/8/17
31	Oprava chyb	6 days	Thu 9/7/17	Fri 9/15/17
32	**Akceptace**	**9.5 days**	**Fri 9/15/17**	**Thu 9/28/17**
33	UAT 1.0	1.5 days	Fri 9/15/17	Mon 9/18/17
34	CR - Ruční změna samplingu	6 days	Mon 9/18/17	Tue 9/26/17
35	CR - Změna šablony testů	4 days	Mon 9/18/17	Fri 9/22/17
36	UAT 1.1	1 day	Tue 9/26/17	Wed 9/27/17
37	Systémová specifikace	1 day	Wed 9/27/17	Thu 9/28/17
38	Release	1 day	Thu 9/28/17	Fri 9/29/17

Obrázek 3.3: Gantt fáze Testování, Akceptace a Release metodiky Etnetera

dokumentace zpropagovat i v hotovém kódu. Díky tomu zapracování trvá déle, než předchozí požadavky. Požadavek na ruční změnu samplingu jsem odhadl na 6 md, protože vyžaduje úpravu zásah do existujícího kódu, což je složitější, než požadavek na změnu šablony testů, kde kód spíše přidávám, než měním (4 md).

Pro zjednodušení předpokládám, že druhé UAT neodhalí chyby v nově implementovaných požadavcích. Díky zkušenosti z minulého testování se testování zrychlí o 0,5 md. Na závěr, po akceptaci klientem vyhotoví senior systémovou specifikaci, kde rozšíří analýzu u seznam změnových požadavků. Etapa akceptace klientem zabrala 9,5 md.

Posledním bodem je release do produkce během posledního dne projektu. Projekt končí v pátek 29. září, pracovní den před deadline 2. října.

3.5 Simulace SI1/2

Průběh simulace je znázorněn na Ganttových diagramech v jednotlivých podkapitolách a probíhá podle výstupu popsaném na konci kapitoly o kurzu Softwarové inženýrství II 2.5.2, který zopakuji na začátku každé podkapitoly dané iterace. Pro připomenutí, metodika se vyznačuje iterativním přístupem a opírá se o dokumentaci v UML jazyce. Jedná se o zjednodušený Unified Process.

V následujících kapitolách rozeberu postupně jednotlivé iterace, odůvodním časové odhady a rozeberu vliv externích událostí na průběh projektu. Projektový tým zahrnuje šest lidí: PM, 3 vývojáři, BA a QA. K vývojářům a PM jsem přidal BA do fáze analýzy a QA, jenž se podílí rovněž na analýze a provádí průběžné testování během vývoje.

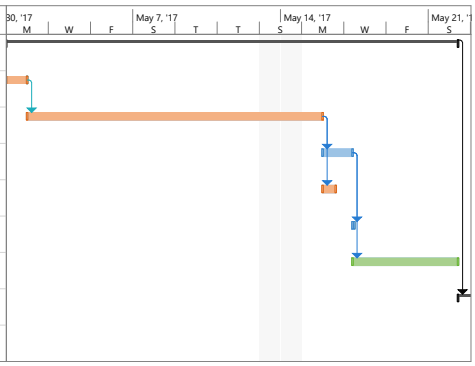
3.5.1 1. iterace

První iterace se nese ve znamení pochopení činnosti zákazníka a analýzy. Převažuje analýza a zahájení práce na návrhu. Výstupem jsou následující body:

- zadávací specifikace a wireframes.
- analýza business procesů *(Activity diagram)*
- stavové diagramy entit *(State Machine diagram)*
- testovací plán

Prvním bodem v Ganttově diagramu je Základní specifikace FP, která trvá 1 md. Tento bod zahrnuje meeting se zákazníkem, během kterého si BA vytvoří představu o funkčních požadavcích na úrovni 3.2.1.

ID	Task Name	Duration	Start	Finish
1	**1. iterace**	**15.25 days**	**Mon 5/1/17**	**Mon 5/22/17**
2	Základní specifikace FP	1 day	Mon 5/1/17	Mon 5/1/17
3	Specifikace FP	10 days	Tue 5/2/17	Mon 5/15/17
4	Wireframy	1.25 days	Tue 5/16/17	Wed 5/17/17
5	Business procesy	0.75 days	Tue 5/16/17	Tue 5/16/17
6	Stavový diagram	0.25 days	Wed 5/17/17	Wed 5/17/17
7	Testovací plán	3 days	Wed 5/17/17	Mon 5/22/17
8	**2. iterace**	**43.75 days**	**Mon 5/22/17**	**Thu 7/20/17**
25	**3. iterace**	**55 days**	**Fri 7/21/17**	**Thu 10/5/17**



Obrázek 3.4: Gantt 1. iterace metodiky SI1/2

Po vymezení základních přestav následuje podrobná specifikace FP s odhadem 10 md. Během těchto dvou týdnů se tvoří detailní zadání, BA se doptává na hlubší souvislosti a vazby jednotlivých požadavků, protože soupis funkčních požadavků z předešlého bodu není dostatečný pro analýzu a návrh řešení.

Po klarifikaci požadavků začnu pracovat souběžně na wireframech a modelování business procesů. Na základě funkčních požadavků jsem napočítal 20 potřebných WF. Vyzkoušel jsem si vytvořit tři WF v programu Balsamiq a tvorba jednoho mi trvala 30 minut. Celková náročnost vychází na 1.25 md (20 * 30 minut). Jedná se o statické wireframy bez možnosti editace formulářových prvků, vyskakování hlášek při událostech a zobrazení modálních oken

na akce. Tento koncept je dostatečný, protože simuluji iterativní metodiku. Jednotlivé WF jsou ovšem průchozí skrze záložky a tlačítka v menu. Zmíněné WF jsou k nahlédnutí v příloze na CD.

Identifikoval jsem čtyři relevantní procesy, v němž figuruje systém SOLO. 1. tvorba úlohy, 2. tvorba testu, 3. vyhodnocení testu a 4. kategorizace úlohy. Pečlivá dokumentace jednoho procesu pomocí Activity Diagramu zabere 1 h + 1 h pochopení domény zákazníka v daném procesu. Pečlivostí myslím granularitu diagramů s případným rozdělením na podprocesy, zanesení výjimek, dodržení notace a hezké formátování. Čerpám z vlastní zkušenosti, kde jsem modeloval například proces nákupu služeb a zpracování platby. Celkově 1 md (4 * 2 h).

Jediným stavovým diagramem je přechodová funkce úlohy. Za předpokladu hotové textové specifikace stavů, přechodových pravidel, omezení na systémová role a vazbu na přiřazení člověka zabere tvorba diagramu 2 hodiny. Stejně jako u aktivity diagramu, čerpám z vlastní pracovní zkušenosti tvorby stavového diagramu licence o 7 stavech, kde těžší než samotné stavy a přechody je definice systémových požadavků a přechodových omezení.

Testovací plán trvá vytvořit 3 md dle 3.3.1.4. Pro úplnost, vývojová prostředí volím testovací a stage. Testovací je interní pro týmového QA. Stage slouží pro UAT na konci 2. a 3. iterace.

Náročnost první iterace vychází při paralelizaci na 15,25 md, tedy tři týdny a jeden den. Během iterace se neobjevily žádné změnové požadavky a překážka v podobě pozdního nástupu juniorního vývojáře neměla na projekt vliv, protože jeho zapojení nebylo během analýzy potřeba. Na konci iterace mám slušnou představu o očekávaném fungování systému.

3.5.2 2. iterace

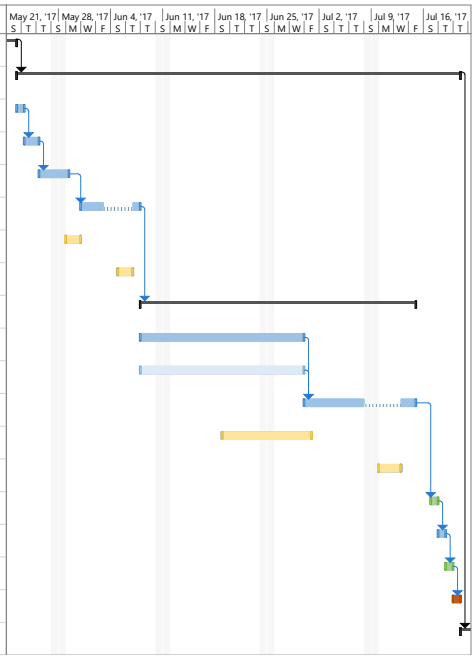
Druhá iterace vychází z analýzy vytvořené během iterace první. Převažuje návrh, doplnění analýzy a zahájení implementace. Součinnost zákazníka není intenzivně vyžadována jako v iteraci předchozí. Během druhé iterace vytvářím návrh řešení a implementuji prototyp. Výstupy druhé iterace:

- návrh logické architektury *(Package, Class diagram)*
- databázový model *(Class diagram)*
- implementace prototypu

Návrh logické architektury je první úkol 2. iterace a zahrnuje vymyšlení třívrstvé architektury, hrubou definici rozhraní SOLO s MensaLide a mezi SOLO FE a BE, práce na 1 md.

Navazující položky volba technologií, tvorba databázového schéma a návrh rozhraní kopíruje časový odhad ze sekce společných aspektů simulace 3.3.1.2.

ID	Task Name	Duration	Start	Finish
1	**1. iterace**	**15.25 days**	**Mon 5/1/17**	**Mon 5/22/17**
8	**2. iterace**	**43.75 days**	**Mon 5/22/17**	**Thu 7/20/17**
9	Architektura	1 day	Mon 5/22/17	Tue 5/23/17
10	Technologie	2 days	Tue 5/23/17	Thu 5/25/17
11	Databáze	2 days	Thu 5/25/17	Mon 5/29/17
12	API	4 days	Wed 5/31/17	Wed 6/7/17
13	CR - více odpovědí	2 days	Mon 5/29/17	Tue 5/30/17
14	CR - váhová obtížnost	2 days	Mon 6/5/17	Tue 6/6/17
15	**Implementace prototypu**	**27 days**	**Thu 6/8/17**	**Fri 7/14/17**
16	FP 3 - Databáze a správa úloh	16 days	Thu 6/8/17	Thu 6/29/17
17	FP 4 - Správa uživatel a rolí	16 days	Thu 6/8/17	Thu 6/29/17
18	FP 1 - Tvorba úloh a kolaborační flow	8 days	Fri 6/30/17	Fri 7/14/17
19	Klient mimo	10 days	Mon 6/19/17	Fri 6/30/17
20	CR - více odpovědí II.	3 days	Mon 7/10/17	Wed 7/12/17
21	UAT 0.1	1 day	Mon 7/17/17	Mon 7/17/17
22	Oprava chyb	1 day	Tue 7/18/17	Tue 7/18/17
23	UAT 0.2	1 day	Wed 7/19/17	Wed 7/19/17
24	Release	1 day	Thu 7/20/17	Thu 7/20/17
25	**3. iterace**	**55 days**	**Fri 7/21/17**	**Thu 10/5/17**



Obrázek 3.5: Gantt 2. iterace metodiky SI1/2

Volba technologií zabere 2 md, stejnou dobu modeluji databázi a 4 md trvá seniorovi detailní návrh API mezi dílčími aplikacemi.

Z hlediska harmonogramu se během návrhu rozhraní vyskytnou první dva změnové požadavky (více odpovědí a váhová obtížnost). Protože ještě nezačala implementace, stojí mě zapracování požadavků zatím pouze změnu analýzy (specifikace, wireframy, testovací plán) a návrhu (databázový model, API). Zapracování jednoho požadavku trvá 2 md, definice API se tedy protáhne o 4 md, a tím se posouvá začátek práce na prototypu.

Prototypem myslím první část funkční aplikace, kterou bude moci zákazník používat a jež bude dále rozvinutelný. Nejedná se o koncept jako u throw-away prototyping, kde prototyp slouží jako ukázka funkcionality, která se posléze zahodí a přepíše „pořádně“. Na prototyp je nahlíženo jako na první softwarový inkrement.

Implementace zahrnuje základ aplikace FP 3 a 4 spolu s FP 1 bez příloh a diskusního vlákna. Reálně se jedná o pokrytí 1/3 databáze, ekvivalentně 20 zdrojů. Po aplikaci formule pro výpočet náročnosti implementace získám 40 md. Během implementace nejdříve v 8. a 9. týdnu přestane zákazník komunikovat. Tato skutečnost vývoj nebrzdí, protože se nacházíme na začátku implementace po ukončení obsáhlé analýzy a návrhu. Naopak změnový požadavek (více odpovědí II.) v 11. týdnu vývoj ovlivní. Jedná se o již druhou změnu formátu odpovědí a na rozdíl od první změny je v tento moment část související funkcionality implementována. Z tohoto důvodu je zapracování po-

žadavku o 1 md delší, než před začátkem práce na prototypu.

Po implementace prototypu přechází SOLO k UAT testování, jenž podle definovaného vzorce odhalí řadu chyb. Protože testujeme 1/3 testovacích scénářů (12 z 35), stačí na UAT 1 md. Oprava chyb vyžaduje 2 md práce, mám ale k dispozici více vývojářů a vlivem paralelizace stáhnu čas na polovinu. Druhé UAT již chyby neodhalí a následný den jsem schopen nasadit do produkce.

Druhá iterace trvá 43,75 dní. Nejvyšší vytíženost vykazuje seniorní a posléze juniorní vývojáři. Během iterace přišly tři změnové požadavky, dva v době, kdy jsem pracoval na návrhu a úpravy se tedy omezily na podpůrnou dokumentaci. Třetí změnový požadavek zbrzdil vývoj, protože se týkal funkcionality, která byla shodou okolností v rámci prototypu implementována. Při volbě odlišného scope prototypu by požadavek neměl tak negativní dopad. Nutno dodat, že zvolený scope prototypu dává dobrý smysl, neb FP 3 a 4 tvoří základ aplikace a FP 1 je stěžejní funkcionalitou SOLO.

3.5.3 3. iterace

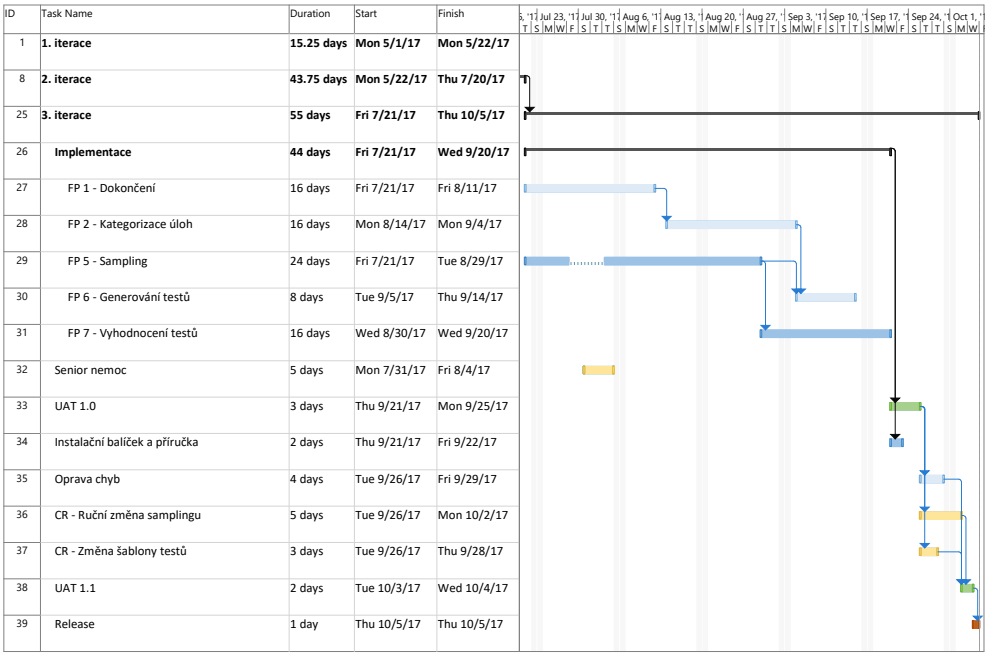
Během poslední iterace převažuje implementace a dokončení návrhu. Výstupem jsou tyto náležitosti:

- instalační balíček
- instalační příručka *(Deployment diagram)*
- vygenerovaná dokumentace ze zdrojových kódů
- zdrojové kódy aplikace

V poslední iteraci zbývá dokončit FP 1 a nově implementovat FP 2, čímž tým dokončí modul úloh. K tomu je potřeba implementovat modul testů, tedy FP 5, FP 6 a FP 7. Hrubý čas implementace zbylých 40 zdrojů odpovídá 80 md. Rozdělením práce mezi 3 vývojáře se dostanu na polovinu. Problém je, že ve 14. týdnu vývoje onemocní seniorní vývojář a odpracuje pouze 1 md. Důsledkem toho trvá implementace 44 md.

UAT testování pokrývá kromě zbylých 23 testovacích scénářů i opakované testování již nasazené části aplikace (12 scénářů). Prakticky tedy testujeme celou aplikaci, což si vyžádá 3 md testování (3x více scénářů, než po implementaci prototypu).

Stejně jako v předešlé iteraci, i zde UAT testování odhalilo chyby. Podle stanoveného klíče vychází oprava na 4 md. Na rozdíl od iterace minulé nemohu opravy paralelizovat, protože spolu s UAT testováním FP 5 a 6 přijde zákazník se změnovými požadavky (ruční sampling, šablony testů). Tím, že požadavky přijdou najednou, mohu spojit jejich vykomunikování se zákazníkem. Moc času ale neušetřím, protože stojím na konci implementace a zasahuji do hotové aplikace. Zapracování nechám na volných juniorech a mezera před druhým kolem testování se protáhne na 5 md.



ID	Task Name	Duration	Start	Finish
1	**1. iterace**	**15.25 days**	**Mon 5/1/17**	**Mon 5/22/17**
8	**2. iterace**	**43.75 days**	**Mon 5/22/17**	**Thu 7/20/17**
25	**3. iterace**	**55 days**	**Fri 7/21/17**	**Thu 10/5/17**
26	**Implementace**	**44 days**	**Fri 7/21/17**	**Wed 9/20/17**
27	FP 1 - Dokončení	16 days	Fri 7/21/17	Fri 8/11/17
28	FP 2 - Kategorizace úloh	16 days	Mon 8/14/17	Mon 9/4/17
29	FP 5 - Sampling	24 days	Fri 7/21/17	Tue 8/29/17
30	FP 6 - Generování testů	8 days	Tue 9/5/17	Thu 9/14/17
31	FP 7 - Vyhodnocení testů	16 days	Wed 8/30/17	Wed 9/20/17
32	Senior nemoc	5 days	Mon 7/31/17	Fri 8/4/17
33	UAT 1.0	3 days	Thu 9/21/17	Mon 9/25/17
34	Instalační balíček a příručka	2 days	Thu 9/21/17	Fri 9/22/17
35	Oprava chyb	4 days	Tue 9/26/17	Fri 9/29/17
36	CR - Ruční změna samplingu	5 days	Tue 9/26/17	Mon 10/2/17
37	CR - Změna šablony testů	3 days	Tue 9/26/17	Thu 9/28/17
38	UAT 1.1	2 days	Tue 10/3/17	Wed 10/4/17
39	Release	1 day	Thu 10/5/17	Thu 10/5/17

Obrázek 3.6: Gantt 3. iterace metodiky SI1/2

Závěrečné UAT probíhá rychleji, protože prototyp testuji prakticky počtvrté a scénáře se zrychlily. Případné drobné chyby opravuji operativně a po 2 md je celá aplikace připravena na release.

Třetí iterace zabrala 55 md a byla nejdelší. Komplikace v podobě výpadku seniorního vývojáře a příchod změnových požadavků posunul harmonogram o 2 týdny. Release se uskutečnil 5. října a vzniklo zpoždění tří pracovních dní oproti domluvenému termínu 2. října.

3.6 Avast

Metodika Avastu je vzhledem k simulaci nejkomplikovanější. Je stavěna na produktový vývoj o více autonomních týmech, kdežto ukázkový projekt předpokládá vývoj projektový o jednom týmu. Projektový tým se skládá z PdM, vývojářů, QA a UX.

Vývojáři fungují kompaktně a v harmonii s agilním manifestem dělají naprosto všechnu práci. Není zde oddělení kompetencí seniorního a juniorních vývojářů. Vývojáři společně nesou označení DevTeam.

Podle metodiky rozdělím vývoj na sérii čtyřtýdenních sprintů, kde každý sprint je zakončen nasazením do produkce. Pro zjednodušení vypouštím nasazování beta verzí každý týden, jelikož simuluji na jednom týmu a ve výsledku by polovina týmu nedělala nic jiného, než že by nasazovala nebo se na release připravovala.

Nasazování probíhá v posledním týdnu každého sprintu. Představa je, že v pondělí začne QA testovat funkcionalitu nového release, zatímco DevTeam operativně nahlášené chyby opravuje. Po dokončení oprav je aplikace předána na UAT testování a v pátek téhož týdne se nasazuje. Je možné, že mezi koncem UAT testování a releasem vznikne časová mezera. Tato mezera je vítaná, jedná se o nárazník kvůli komplikacím a při jeho nepotřebnosti pokračuje DevTeam v implementaci.

Protože neprobíhá detailnější návrh a analýza jako u iterativní či vodopádové metodiky, zvyšuji implementační čas vycházející ze vztahu 3.1 o 30 %. Navýšení pokrývá návrh s analýzou za běhu a refactoring spojený s přidáváním nové funkcionality. Navíc odhady zaokrouhluji nahoru na celé md, protože agilní metodika plánuje po dnech a s drobnějším dělením by se obtížně pracovalo. Vztah pro výpočet náročnosti implementace je tedy:

$$2 * n * 1.3 * \textit{výkon týmu} \quad md \tag{3.4}$$

Ačkoli tým nezpracovává konkrétní testovací plán, při pozorování v Avastu jsem si všiml, že QA téměř nepustí do release implementaci s chybou. Zajetý release plan obsahuje detailní checklist procedur předcházející nasazení. QA raději upřednostní zmenšení funkcionality release před vypuštěním něčeho, co funguje napůl. Nízkou chybovost doplňuje vysoká míra unit, integračních a automatizovaných UI testů. Z těchto důvodu zanedbávám odhalení dodatečných chyb při UAT testování, kterému předchází právě testování interní.

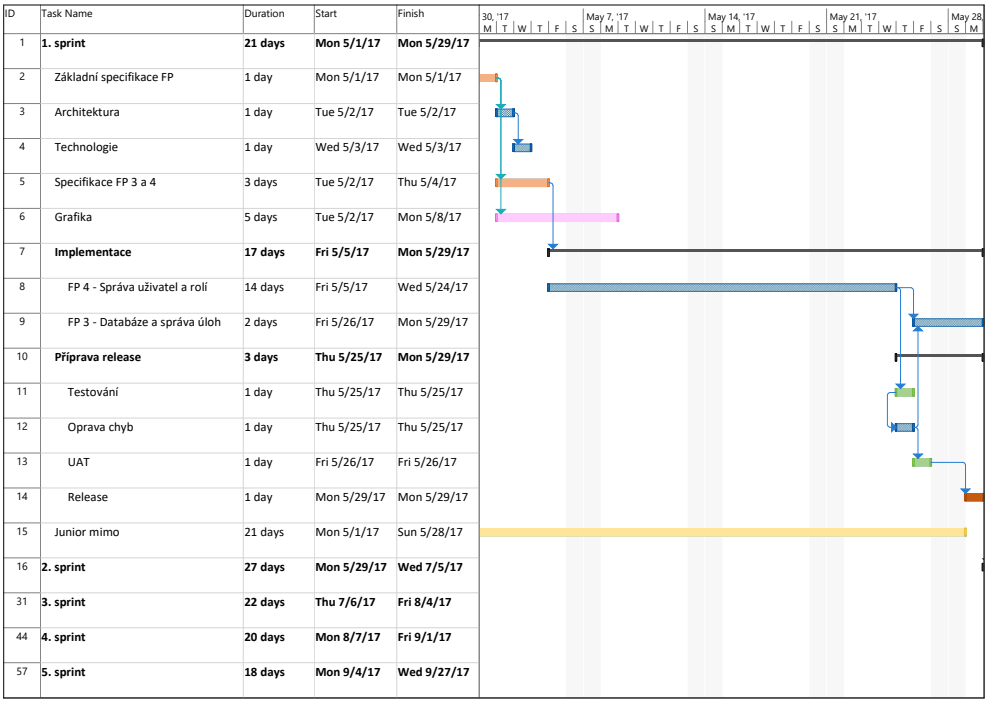
3.6.1 1. sprint

Úvodní sprint začíná sběrem požadavků jako u předešlých metodik. Po prvním dni má tým rámcově jasno, co bude implementovat. Následuje paralelní souběh práce DevTeamu, PdM a UX. Na návrhu základní architektury a volbě technologií se podílí oba vývojáři po dobu 2 md. Třetí vývojář má opožděný nástup až v 5. týdnu vývoje a jeho absence se ještě projeví při implementaci.

Zatímco DevTeam vytváří skeleton aplikace, PdM operativně se zákazníkem komunikuje zadání prvních požadavků. Konkrétně se jedná o FP 3 a FP 4, tedy základ aplikace. Komunikaci, odhady a plánování zaberou 3 md dle 3.3.1.1. Poslední souběžnou činností je tvorba grafiky od UX designéra v náročnosti 5 md. Grafická podoba FE je mimo zadání jedinou hmatatelnou dokumentací.

Jakmile PdM dokončí specifikaci prvních požadavků, DevTeam začíná implementovat. Práce na grafice pokračuje souběžně a DevTeam nebrzdí, neb z počátku je potřeba implementovat databázi a aplikační logiku. DevTeam má úroveň výkonu 150 % (oproti výkonu samotného seniora). Prvním předmětem implementace je FP 4 trvající 14 md.

Protože bez hotového FP 4 by nebylo co zákazníkovi předvést, musím posunout začátek testování z pondělí na čtvrtek. Interní testování s opravou



ID	Task Name	Duration	Start	Finish
1	**1. sprint**	**21 days**	**Mon 5/1/17**	**Mon 5/29/17**
2	Základní specifikace FP	1 day	Mon 5/1/17	Mon 5/1/17
3	Architektura	1 day	Tue 5/2/17	Tue 5/2/17
4	Technologie	1 day	Wed 5/3/17	Wed 5/3/17
5	Specifikace FP 3 a 4	3 days	Tue 5/2/17	Thu 5/4/17
6	Grafika	5 days	Tue 5/2/17	Mon 5/8/17
7	**Implementace**	**17 days**	**Fri 5/5/17**	**Mon 5/29/17**
8	FP 4 - Správa uživatel a rolí	14 days	Fri 5/5/17	Wed 5/24/17
9	FP 3 - Databáze a správa úloh	2 days	Fri 5/26/17	Mon 5/29/17
10	**Příprava release**	**3 days**	**Thu 5/25/17**	**Mon 5/29/17**
11	Testování	1 day	Thu 5/25/17	Thu 5/25/17
12	Oprava chyb	1 day	Thu 5/25/17	Thu 5/25/17
13	UAT	1 day	Fri 5/26/17	Fri 5/26/17
14	Release	1 day	Mon 5/29/17	Mon 5/29/17
15	Junior mimo	21 days	Mon 5/1/17	Sun 5/28/17
16	**2. sprint**	**27 days**	**Mon 5/29/17**	**Wed 7/5/17**
31	**3. sprint**	**22 days**	**Thu 7/6/17**	**Fri 8/4/17**
44	**4. sprint**	**20 days**	**Mon 8/7/17**	**Fri 9/1/17**
57	**5. sprint**	**18 days**	**Mon 9/4/17**	**Wed 9/27/17**

Obrázek 3.7: Gantt 1. sprint metodiky Avast

chyb trvá kombinovaně 1 md. V pátek úspěšně klient akceptuje release a do produkce se první verze aplikace dostává se zpožděním jednoho dne, v pondělí. Ve stejný den nastoupil do projektu poslední vývojář. Během UAT a release DevTeam započal práci na FP 3.

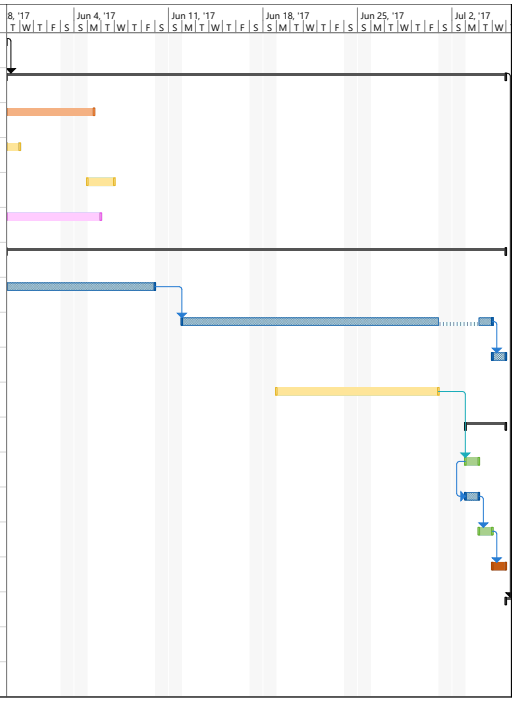
První sprint trval 21 dní a došlo ke zpoždění o jeden den, protože tým nemohl nasadit, dokud neimplementoval alespoň jednu kompletní funkcionalitu. Příčinnou zdržení byla absence jednoho z vývojářů. Po sprintu má aplikace hotovou správu uživatel a započatou práci na modulu úloh.

3.6.2 2. sprint

Během druhého sprintu je DevTeam v plném počtu. Sprint začíná pro PdM specifikací FP 1 a 2 v celkovém objemu 4,5 md. Tyto požadavky logicky navazují na FP 3. Během specifikace přijde klient s prvním změnovým požadavkem (více odpovědí). Vzhledem k tomu, že teprve funkcionalitu podrobně specifikujeme a nezačala na ní implementace, požadavek nijak nenaruší průběh projektu a je zahrnut do času specifikace FP 1 a 2. Stejně jako v přechozím sprintu, i na začátku tohoto probíhá tvorba grafiky, tentokrát pro příslušné FP 1 a 2. V druhém týdnu sprintu přijde další změnový požadavek (váhová obtížnost). Požadavek vyvstal až při dokončování specifikace a práce na grafice. Jeho zapracování si tedy vyžádá delší čas, než předešlý požadavek, ale

naštěstí DevTeam ještě nezačal na požadavku pracovat, takže změny se kódu netýkají.

ID	Task Name	Duration	Start	Finish
1	**1. sprint**	**21 days**	**Mon 5/1/17**	**Mon 5/29/17**
16	**2. sprint**	**27 days**	**Mon 5/29/17**	**Wed 7/5/17**
17	Specifikace FP 1 a 2	4.5 days	Tue 5/30/17	Mon 6/5/17
18	CR - více odpovědí	1 day	Mon 5/29/17	Tue 5/30/17
19	CR - váhová obtížnost	2 days	Mon 6/5/17	Tue 6/6/17
20	Grafika	5 days	Tue 5/30/17	Mon 6/5/17
21	**Implementace**	**27 days**	**Tue 5/30/17**	**Wed 7/5/17**
22	FP 3 - dokončení	9 days	Tue 5/30/17	Fri 6/9/17
23	FP 1 - Tvorba úloh a flow	16 days	Mon 6/12/17	Tue 7/4/17
24	FP 2 - Kategorizace úloh	1 day	Wed 7/5/17	Wed 7/5/17
25	Klient mimo	10 days	Mon 6/19/17	Fri 6/30/17
26	**Příprava release**	**3 days**	**Mon 7/3/17**	**Wed 7/5/17**
27	Testování	1 day	Mon 7/3/17	Mon 7/3/17
28	Oprava chyb	1 day	Mon 7/3/17	Mon 7/3/17
29	UAT	1 day	Tue 7/4/17	Tue 7/4/17
30	Release	1 day	Wed 7/5/17	Wed 7/5/17
31	**3. sprint**	**22 days**	**Thu 7/6/17**	**Fri 8/4/17**
44	**4. sprint**	**20 days**	**Mon 8/7/17**	**Fri 9/1/17**
57	**5. sprint**	**18 days**	**Mon 9/4/17**	**Wed 9/27/17**



Obrázek 3.8: Gantt 2. sprint metodiky Avast

DevTeam na začátku sprintu dokončuje FP 3. V minulém sprintu stihli dva vývojáři odpracovat 2 md. Pokud by v práci pokračovali, trvalo by jim to 14 dní a udělali tedy 1/7 práce. Kompletní DevTeam tedy potřebuje 9 md (odhad dle 3.4 * 6/7 práce). Končí v pátek druhého týdne. Následující týden začíná DevTeam pracovat na FP 1. V navazujícím týdnu, kdy se má nasazovat, ovšem přestane zákazník komunikovat, protože odletěl na neohlášenou dovolenou. Tým neví, jak dlouho bude absence trvat. Protože má část požadavků specifikovaných dopředu, pokračuje v implementaci toho, co ví (FP 1 a 2) a doufá, že se klient do té doby ozve.

Klient začne komunikovat po 14 dnech. Za tuto dobu stihl tým téměř implementovat FP 1 s náročností 16 md. Jakmile se klient vrátil, začíná proces nasazování. První den se stihne interně otestovat nová funkcionalita a zapracovat opravy. Chyb je nalezeno více kvůli větší části implementace, ale zároveň chyby odbavuje více vývojářů. Z těchto důvodů trvá testování na straně dodavatele opět jeden den. Aplikace ještě není tak velká, aby došlo ke zdržení u UAT nebo release (obě akce znovu po 1 md). Nasazeno je úspěšně za tři dny.

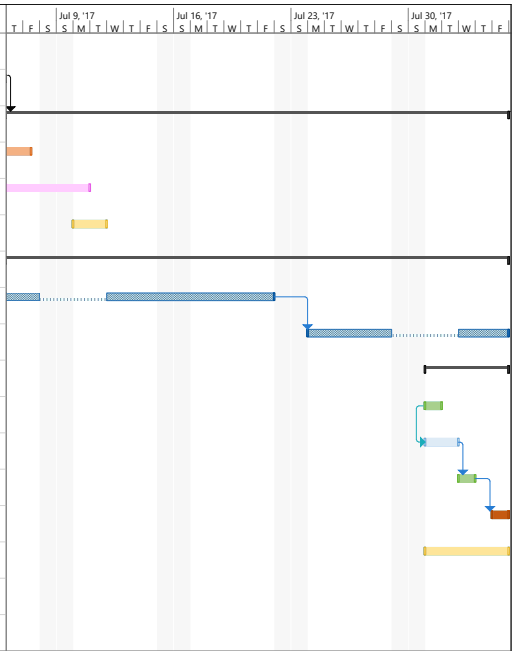
Druhý sprint se protáhl na 27 dní, což odpovídá zhruba o třetinu. Na začátku přišel zákazník se změnovými požadavky, které ale vývoj neovlivnily v takové míře, protože DevTeam na funkcionalitě tou dobou ještě nepracoval

a změny se týkaly pouze zadání a grafiky. Ke konci sprintu přestal zákazník komunikovat, a to vedlo tým k improvizaci. Vzhledem k tomu, že tým zadání komunikuje s relativním předstihem, mohl po dobu nepřítomnosti implementovat a klient se vrátil shodou okolností dřív, než došlo zadání. Nasazování začalo v pondělí o dva týdny později, než bylo plánováno. Sprint skončil ve středu ten samý týden (6. týden sprintu).

3.6.3 3. sprint

Plánovaná délka třetího sprintu je 22 dní. Ke standardním čtyřem týdnům přidám čtvrtek a pátek z týdne, kde končil sprint předešlý kvůli zarovnání harmonogramu. Obdobně jako na začátku předešlých sprintů, i zde PdM zjišťuje požadavky. Tentokrát se jedná pouze o FP 5, a proto samotné zadání i grafika trvají kratší dobu (dříve se zjišťovaly vždy dva požadavky).

ID	Task Name	Duration	Start	Finish
1	**1. sprint**	**21 days**	**Mon 5/1/17**	**Mon 5/29/17**
16	**2. sprint**	**27 days**	**Mon 5/29/17**	**Wed 7/5/17**
31	**3. sprint**	**22 days**	**Thu 7/6/17**	**Fri 8/4/17**
32	Specifikace FP 5	1.5 days	Thu 7/6/17	Fri 7/7/17
33	Grafika	3 days	Thu 7/6/17	Mon 7/10/17
34	CR - více odpovědí II.	2 days	Mon 7/10/17	Tue 7/11/17
35	**Implementace**	**22 days**	**Thu 7/6/17**	**Fri 8/4/17**
36	FP 2 - dokončení	10 days	Thu 7/6/17	Fri 7/21/17
37	FP 5 - Sampling	8 days	Mon 7/24/17	Fri 8/4/17
38	**Příprava release**	**5 days**	**Mon 7/31/17**	**Fri 8/4/17**
39	Testování	1 day	Mon 7/31/17	Mon 7/31/17
40	Oprava chyb	2 days	Mon 7/31/17	Tue 8/1/17
41	UAT	1 day	Wed 8/2/17	Wed 8/2/17
42	Release	1 day	Fri 8/4/17	Fri 8/4/17
43	Senior mimo	5 days	Mon 7/31/17	Fri 8/4/17
44	**4. sprint**	**20 days**	**Mon 8/7/17**	**Fri 9/1/17**
57	**5. sprint**	**18 days**	**Mon 9/4/17**	**Wed 9/27/17**



Obrázek 3.9: Gantt 3. sprint metodiky Avast

DevTeam mezitím dokončuje práci na FP 2, na které zbývá odpracovat 10 z 11 md. V pondělí druhého týdne přijde změnový požadavek (více odpovědí II.). Jedná se již o implementovanou funkcionalitu, tím pádem během 2 md se kromě zadání a grafiky mění i kód, aby byl požadavek připraven do dalšího release. Z tohoto důvodu je na chvíli pozastavena práce na FP 2. Po odbavení požadavku dokončí DevTeam FP 2 a stíhá započít práci na FP 5. V týdnu přípravy nové verze onemocní senior, a proto oslabenému DevTeamu (dva junioři) trvá zapracování chyb z testování déle. Díky tomu, že testovat

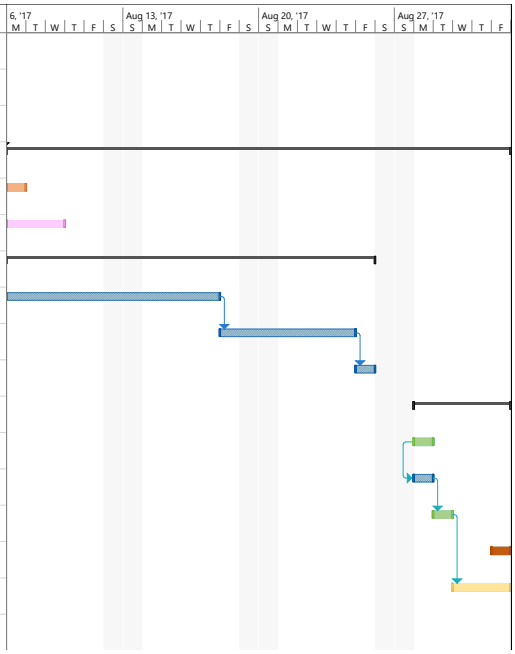
začínáme v pondělí a release je naplánován na pátek, využijeme část nárazníkového polštáře a termín release zůstává neovlivněn. Do konce sprintu pokračují junioři na implementaci FP 5.

Třetí sprint se jako první vyhnul posunutí termínu nasazení a trval předpokládaných 22 dní. Podařilo se dokončit FP 2, což znamená hotový modul úloh a začala se práce na FP 5, stavebním kameni modulu testů. Na FP 5 se odpracovalo 8 dní, ale z toho 5 dní ve 3 lidech a 3 dny ve 2 lidech s omezenou podporou seniora. Pracovní ekvivalent je tedy 7 md DevTeamu.

3.6.4 4. sprint

Ve čtvrtém sprintu je DevTeam opět v plné síle. PdM a UX zbývá připravit podklady pro FP 6 a 7. Jedná se sice o dva požadavky, nicméně jsou to požadavky menšího rozsahu, a z toho důvodu trvá specifikace i s grafikou 3 md.

ID	Task Name	Duration	Start	Finish
1	**1. sprint**	**21 days**	**Mon 5/1/17**	**Mon 5/29/17**
16	**2. sprint**	**27 days**	**Mon 5/29/17**	**Wed 7/5/17**
31	**3. sprint**	**22 days**	**Thu 7/6/17**	**Fri 8/4/17**
44	**4. sprint**	**20 days**	**Mon 8/7/17**	**Fri 9/1/17**
45	Specifikace FP 6 a 7	1 day	Mon 8/7/17	Mon 8/7/17
46	Grafika	3 days	Mon 8/7/17	Wed 8/9/17
47	**Implementace**	**15 days**	**Mon 8/7/17**	**Fri 8/25/17**
48	FP 5 - dokončení	9 days	Mon 8/7/17	Thu 8/17/17
49	FP 6 - Generování testů	5 days	Fri 8/18/17	Thu 8/24/17
50	FP 7 - Vyhodnocení testů	1 day	Fri 8/25/17	Fri 8/25/17
51	**Příprava release**	**5 days**	**Mon 8/28/17**	**Fri 9/1/17**
52	Testování	1 day	Mon 8/28/17	Mon 8/28/17
53	Oprava chyb	1 day	Mon 8/28/17	Mon 8/28/17
54	UAT	1 day	Tue 8/29/17	Tue 8/29/17
55	Release	1 day	Fri 9/1/17	Fri 9/1/17
56	CR - Ruční změna samplingu	3 days	Wed 8/30/17	Fri 9/1/17
57	**5. sprint**	**18 days**	**Mon 9/4/17**	**Wed 9/27/17**



Obrázek 3.10: Gantt 4. sprint metodiky Avast

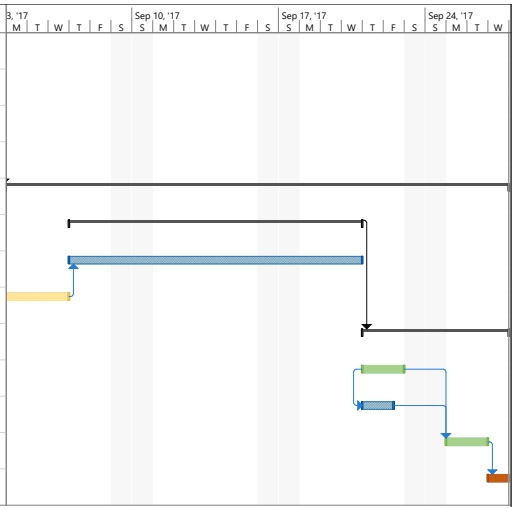
DevTeamu zbývá odpracovat 9 md (16 md odhad - 7 md v 3. sprintu) na FP 5. Jako navazující požadavek upřednostňuje FP 6 před FP 7, protože se jedná o menší celek (5 md), který se stihne kompletně implementovat ještě v aktuálním sprintu. Před přípravou release tomu tým dostojí a začne ještě pracovat na FP 7 po dobu 1 md. Po UAT přijdou změnové požadavky jak na funkčnost samplingu, tak na generování testů. Jedná se o netriviální požadavky a DevTeam stihne do konce sprintu zapracovat pouze první z nich. Implementace ale není otestována a bude předmětem až dalšího nasazení.

Čtvrtý sprint trval přesně 4 týdny. Tým nenarazil v prvních třech týdnech na komplikace, což se odrazilo v dokončení požadavků FP 5 a 6. To se týmu vyplatilo, protože požadavky prošly UAT testováním, kde si klient uvědomil, že by chtěl změnit jejich funkčnost a tým je schopen připomínky zapracovat během příštího, plánovaného posledního sprintu.

3.6.5 5. sprint

Závěrečný sprint se obešel bez specifikace požadavků. Tým dokončil zadání v minulém sprintu a nyní se pouze operativně komunikuje na denní bázi. Na začátku sprintu nejdříve DevTeam zapracovává druhý změnový požadavek ještě z UAT testování sprintu minulého. Jakmile má hotovo, pokračuje na FP 7, který stihl rozpracovat skoro před dvěma týdny.

ID	Task Name	Duration	Start	Finish
1	**1. sprint**	**21 days**	**Mon 5/1/17**	**Mon 5/29/17**
16	**2. sprint**	**27 days**	**Mon 5/29/17**	**Wed 7/5/17**
31	**3. sprint**	**22 days**	**Thu 7/6/17**	**Fri 8/4/17**
44	**4. sprint**	**20 days**	**Mon 8/7/17**	**Fri 9/1/17**
57	**5. sprint**	**18 days**	**Mon 9/4/17**	**Wed 9/27/17**
58	**Implementace**	**10 days**	**Thu 9/7/17**	**Wed 9/20/17**
59	FP 7 - dokončení	10 days	Thu 9/7/17	Wed 9/20/17
60	CR - Změna šablony testů	3 days	Mon 9/4/17	Wed 9/6/17
61	**Příprava**	**5 days**	**Thu 9/21/17**	**Wed 9/27/17**
62	Testování	2 days	Thu 9/21/17	Fri 9/22/17
63	Oprava chyb	1.5 days	Thu 9/21/17	Fri 9/22/17
64	UAT	2 days	Mon 9/25/17	Tue 9/26/17
65	Release	1 day	Wed 9/27/17	Wed 9/27/17



Obrázek 3.11: Gantt 5. sprint metodiky Avast

V polovině třetího týdne je implementace dokončena. Interní testování celé aplikace začíná s předstihem a vyžádá si více času, než obvyklé testování přírůstku. Automatizované testování a cvik z minulých release nakonec úspěšně ukončuje testování po dvou dnech. Během důkladného testování se narazí i na více chyb než dříve. I tak DevTeam při plném nasazení stíhá opravit do konce třetího týdne.

Tým se s klientem domluvil, že UAT testování může začít již v pondělí čtvrtého týdne sprintu. UAT je stejně jako interní testování podrobnější, protože se jedná o závěrečný release. Aplikace se dostává kompletně do provozu ve středu 27. září, tři pracovní dny před termínem 2. října.

3.7 Shrnutí

Simulace ukázala průběh projektu se stejným zadáním a vnějšími podmínkami za pomocí třech odlišných metodik. Ukázkový projekt jsem navrhnul předem a během simulace ho nijak neměnil. I přes tento aspekt metodiky dokončily projekt překvapivě za přibližně stejnou dobu, jak znázorňuje tabulka 3.2. Konec projektu vychází u všech metodik v rozpětí jednoho týdne. Z tohoto hlediska jsou metodiky srovnatelné.

Tabulka 3.2: Ukončení projektů

Metodika	Typ	Konec
Etnetera	vodopád	29. září (-1 md)
SI1/2	iterativní	5. října (+3 md)
Avast	agilní	27. září (-3 md)

Metodiky Etnetera a SI1/2 se ukázaly být podobné. Vodopádová Etnetera neměla takové zdržení na analýze a návrhu, jak jsem očekával. Implementace začala o 6 pracovních dní později než v případě SI1/2. Vysvětlením je, že Etnetera v praxi netvoří tolik dokumentace, jako se od učebnicového, vodopádového přístupu očekává.

Průběh metodikou Avastu probíhal naprosto odlišně, přesto dosáhl stejného cíle a pomyslný závod vyhrál. Simulace ukázala vysokou křehkost dodržení metodiky, kde vnější okolnosti měly významný dopad na dodržení termínu nebo scope sprintu. Při kratších intervalech nasazení je obtížnější tyto problémy pojmout a dopad je zřetelnější.

Při delším trvání projektu nebo více problémech v druhé polovině realizace očekávám, že by se iterativní metodika dostala do čela, neb u agilního způsoby by se začala projevovat jednak problematika technického dluhu způsobená chybějícím návrhem z konceptuálního pohledu a jednak nepřehlednost vinou absence dokumentace. Na druhou stranu vodopád by trávil více času právě úpravou dokumentace, než metodika iterativní.

Navíc iterativní metodika by ušetřila zavedením interního testování před předáním aplikace k UAT, protože vývojáři by chyby opravovali operativně a nemuseli by čekat na report od klienta. Interní testování jsem nezahrnul, protože na základě kurzu SI1/2 není definované, jak by mělo přesně probíhat.

Ačkoli iterativní metodika přetáhla deadline o 3 dny, z hlediska údržby a budoucího vývoje se jedná o projekt s nejlepšími předpoklady. Vytvořená dokumentace totiž pokrývá aplikaci z několika úhlů pohledu. Navíc model nasazení a aktivity diagramy usnadní zaučení potenciálních nových členů týmu.

Agilní metodika vytváří dokumentaci podle potřeby. Zde je velkou neznámou, kdy nastává ten okamžik, kdy je dokumentace potřeba a jak dlouho trvá její realizace. Neexistence dokumentace vytváří nátlak na dodavatelskou společnost, protože členové týmu se stávají nepostradatelnými díky jedinečné znalosti systému.

Během simulace jsem tiše předpokládal průběh bez vlivu souběžných projektů. Zahrnutí tohoto aspektu by simulaci udělalo značně náročnější a jedná o aspekt mimo rozsah bakalářské práce. Pokud se nad tématem pozastavím, lze si všimnout, že u agilní metodiky je tým stoprocentně koncentrován na projekt po celou jeho dobu. Členové týmu se vyhýbají přepínání kontextu a mohou se plně soustředit na vývoj a vylepšování procesu.

Naopak u vodopádového a iterativního průběhu nastávala časová okna, během kterých nebyli potřeba všichni členi týmu. Analytici se podíleli výhradně na analýzách a juniorní vývojáři byli potřeba jen při implementaci, neb seniorní vývojář navrhoval řešení sám. Tato volná alokace dává prostor buď pro zamyšlení se nad zapojením nepotřebných členů týmu při ostatních aktivitách za cílem zrychlení projektu, nebo využití zdrojů na jiném projektu, kde vzniká kontraproduktivní nutnost přepínání kontextu.

Nespornou výhodou agilní metodiky bylo časté nasazování. Agilní metodika nasadila pětkrát, iterativní dvakrát a vodopádová pouze jednou. Při zkoumání releasů agilní metodiky s nadhledem je vidět, že již v 2. release je implementován stěžejní požadavek FP 1. Zákazník tedy benefituje z aplikace již po dvou měsících vývoje a může ji začít využívat z části k tomu, k čemu potřebuje. Podobná situace nastala i u iterativní metodiky, kde po dvou měsících vyšel funkční prototyp aplikace. Ten nicméně nedosahoval takové funkcionality jako 2. release agilního vývoje. Navíc agilní vývoj vydával update každý měsíc, kdežto u zbylých metodik si zákazník musel počkat až na konec projektu na něco nového (v případě vodopádu na všechno).

To mě přivádí k dalšímu aspektu, který jsem pro zjednodušení vynechal, a tím je migrace dat. Argument je stejný jako u zanedbání vlivu ostatních projektů. Migraci dat by navíc komplikovaly změnové požadavky, jenž pokaždé měnily databázový model aplikace.

Simulace tedy ukázala, že podle časového kritéria dosahují metodiky na středním projektu o délce pěti měsíců při takto stanovených parametrech srovnatelného výsledku.

Kapitola 4

Vyhodnocení

V předcházejících kapitolách studie jsem popsal používané metodiky ve vybraných IT společnostech a metodiku vyučovanou během bakalářského studia na FIT ČVUT. Následovalo provedení simulace některých metodik na ukázkovém projektu. Nyní je čas na vyhodnocení metodik dle výsledků simulace a zvolených kritérií.

4.1 Simulace a srovnávací kritéria

V úvodu práce jsem definoval srovnávací kritéria, podle kterých hodnotím jednotlivé metodiky. Kritérii jsou naplnění cíle (a kvality), doba a náklady. Výstup provedené simulace je pro tato kritéria podkladem. Z důvodu úzké provázanosti srovnávacích kritérií a simulace hodnotím v této části pouze ony metodiky, které simulací prošly.

4.1.1 Cíl

Cíl se podařilo naplnit u všech metodik. Pozornost si tedy zaslouží další složky kritéria, kterými jsou frekvence nasazení nebo reakce na změny.

Z povahy metodiky byla nejvyšší frekvence nasazování u Avastu, kde první nasazení proběhlo již po prvním sprintu. Následoval release každý sprint, kde aplikace byla použitelná od konce sprintu druhého (za necelých 10 týdnů vývoje). Pro srovnání, koncem 12. týdne vývoje nasazovala metodika SI1/2 funkční prototyp. Výhoda častého nasazování Avastu bylo včasnější zapracování změnových požadavků vázaných na zpětnou vazbu od klienta. Navíc zákazník mohl aplikaci začít používat o 3 měsíce dříve, než u vodopádové Etnetery, kde proběhl release až po konci implementace. V případě Etnetery sice vznikl dynamický wireframe model, ten je ale zákazníkem nepoužitelný a navíc může v zákazníkovi vyvolat pocit, že už je vlastně hotovo.

Ačkoli SI1/2 nasadila prototyp ve 3. měsíci projektu, od té doby již znovu nenasadila a zákazník neměl k dispozici update aplikace do skončení projektu,

na rozdíl od Avastu. Nasazování, mimo dřívějšího odkrytí chyb a změn, dodává zákazníkovi jistotu, že se projekt posouvá dopředu, neboť kromě reportů PM vidí reálný výstup. Je potřeba dodat, že projekt trval pouhých pět měsíců, při uvážení například ročního projektu by rozdíl ve frekvenci nasazování a plynoucích benefitů byl ještě markantnější. Celková bilance nasazení je pět pro Avast, dvě pro SI1/2 a jedno pro Etneteru.

S frekvencí nasazení se pojí i reakce na změny, kde opět má přístup Avastu výhodu. Benefit Avastu umocňuje i fakt, že minimalizací dokumentace bylo zapracování požadavků rychlejší, než u zbylých metodik. Podrobná čísla a důvody diskutuji v kritériu nákladů. U metodik Etnetera a SI1/2 hodně zapracování ovlivnilo, zda se změnový požadavek dotknul pouze analýzy, návrhu nebo i započaté, potažmo hotové, implementace.

Zkušenost vývojářů byla na totožné úrovni u všech metodik. V dílčím kritériu kvality tedy hraje role detail a dokumentace návrhu. Ačkoli dokumentace SI1/2 popisuje aplikaci formalizovaně z více úhlů pohledu, plnohodnotný wireframe model u Etnetery pokládám za přínosnější, protože konkrétně ukazuje chování systému na uživatelské úrovni. Klient i dodavatel mají představu, co zákazník chce, respektive dostane. Vývojáři lépe porozumí požadavkům zákazníka a vazbám mezi vrstvami aplikace. Avast se omezil v návrhu na grafiku, které si zákazník všimne nejdříve. Návrh probíhá za běhu a dokumentace se vytváří dle potřeby. Zde je obtížné zachytit, kdy nastává onen moment potřeby.

Každý typ dokumentace dává smysl a i odvážný přístup Avastu prokazatelně funguje, byť klade nároky na zralost týmu. Nejvyšší kvalitu nicméně přisuzuji metodice SI1/2, protože je to právě technická dokumentace, která hraje zásadní roli při dlouhodobém vývoji. Návrh serverové části aplikace dle uživatelského rozhraní v případě Etnetery není doporučený způsob, neb aplikační vrstva má být nezávislá na vrstvě presentační. U Avastu je problém ve zmíněné senioritě a držení znalosti v týmů lidí, což může mít následky při obměně členů týmu.

Tabulka 4.1: Vyhodnocení cíle

Pořadí	1.	2.	3.
Metodika	Avast	SI1/2	Etnetera

I přes kritiku Avastu v otázce dokumentace řadím metodiku v tabulce 4.1 na první místo kvůli frekvenci nasazení a plynoucím benefitům. Dokumentace je minimalistická, ale dostatečná a funkční. Následuje SI1/2, která si vedla o něco lépe než Etnetera v obou aspektech hodnocení. Ačkoli je Etnetera v tomto bodě na posledním místě, není její přístup špatný, jen ostatní přístupy jsou lepší. Hlavní nevýhodou Etnetery je pouze jedno nasazení do produkce.

4.1.2 Doba

Jak jsem uvedl v bezprostředním shrnutí kapitoly simulace, metodiky si z časového hlediska vedly prakticky stejně. Etnetera a Avast dokončily aplikace s časovým předstihem, a v případě SI1/2 nastalo prodlení tří pracovních dní. V tabulce 4.2 shrnuji základní milníky harmonogramu vzhledem ke startu 1. 5. 2017 a předpokládaného konce 2. 10. 2017.

Tabulka 4.2: Milníky projektů

Metodika	Typ	Start vývoje	První release	Konec
Etnetera	vodopád	7. 7.	29. 9.	29. 9. (-1 md)
SI1/2	iterativní	8. 6.	20. 7.	5. 10. (+3 md)
Avast	agilní	5. 5.	29. 5.	27. 9. (-3 md)

Za zmínku stojí datum startu vývoje, od kterého se dá odvinout plánování zdrojů. Dopad měl totiž okamžik i při simulaci, kde metodiky Etnetera a SI1/2 neovlivnila absence juniorního vývojáře během prvních pěti týdnů projektu, protože nebyl při analýze a návrhu potřeba. Datum prvního release vyzdvihuje benefit brzkého představení části aplikace zákazníkovi související s předešlým kritériem.

Tabulka 4.3: Vyhodnocení doby

Pořadí	1.	2.	3.
Metodika	Avast	Etnetera	SI1/2

V konečném důsledku nejsou data startu vývoje a prvního release pro klienta důležitá tak, jako konec projektu. Na termín dodání je často navázán další business zákazníka, a proto je prioritnější. Dřívější částečný release je vítaný, nicméně hotová aplikace je to, na čem z pohledu zákazníka záleží více. Datum začátku vývoje jsem uvedl pro představu a zamyšlením se nad plánováním zdrojů. Nedá se na jeho základě stanovit, zda metodika je špatná nebo dobrá, neb datu předchází u jednotlivých metodik absolutně rozdílné činnosti. Z těchto důvodů hodnotím v tabulce 4.3 metodiky dle termínů dodání.

4.1.3 Náklady

Analýzu nákladů jsem provedl pomocí aplikace MS Project. Protože jsem jednotlivé úkoly během simulace přiřazoval zdrojům, mohu data získat přepnutím pohledu z Ganttova diagramu na vytíženost zdrojů. Tabulka 4.4 ukazuje agregovaná data ze zdrojových MS Project souborů dostupných v příloze studie.

Z dat je vidět, že agilní metodika předčila v předešlém kritériu ostatní metodiky za cenu podstatného zvýšení pracnosti. Důvodem je to, že na vývoji se vždy podílel vývojový tým jako celek (tedy ve třech lidech), kdežto u zbylých metodik se požadavky rozhazovaly mezi seniorního a juniorní vývojáře, a pak

Tabulka 4.4: Přehled zdrojů

Metodika	Role	Počet lidí	Pracnost (md)
Etnetera	PM, 3 Dev, AM, BA, QA	7	254
SI1/2	PM, 3 Dev, BA, QA	6	237
Avast	PdM, 3 Dev, UX, QA	6	334

se stávalo, že někteří vývojáři byli po určitou dobu nevyužití. Navíc u metodik Etnetera a SI1/2 nebyli juniorní vývojáři zapojeni během specifikace, analýzy a návrhu řešení, což je podstatná doba během které mohou být využiti jinak nebo na jiném projektu.

Tabulka 4.5 přináší detailnější rozpad pracnosti. Sloupec analýzy obsahuje i komunikaci specifikace se zákazníkem. Do release jsem zahrnul rovněž čas strávený testováním před nasazením do produkce. Zapracování změnových požadavků nerozčleňuji mezi jednotlivé aktivity, ale uvádím samostatně. Vývoj je sloučením prací všech vývojářů. Nutno podotknout, že odhady zahrnují pouze aktivity zanesené v Ganntových diagramech, které jsou kvůli přehlednosti zjednodušeny o paralelní procesy nemající vliv na trvání projektu (průběžná práce PM, příprava testovacích scénářů QA, nastavení infrastruktury během analýzy atp.). Na Ganttovy diagramy se dá nahlížet jako na representaci kritické cesty projektu.

Tabulka 4.5: Přehled pracnosti (md)

Metodika	Analýza	Vývoj	Release	Změnové požadavky	Celkem
Etnetera	14	208	8	24	254
SI1/2	12	198	12	15	237
Avast	33	272	17	11	334

Závěr z tabulkových výstupů je, že metodika Avastu má nejvyšší náklady na zdroje vlivem stoprocentního vytížení vývojového týmu, spolu s častou komunikací zadání a režijními náklady frekventovaného nasazování. U této metodiky hodně záleží, jak jsou opakované činnosti efektivní a jak se reálně projeví absence dokumentace při vývoji, kterou jsem ve studii nastavil penalizací 30 %.

Metodiky Etnetera a SI1/2 vykazují podobné výsledky. Důvodem je opět to, že SI1/2 produkuje na iterativní metodiku dost dokumentace v porovnání s praxí ořezanou Etneterou, kde je dokumentace na vodopád méně, nýbrž dostatečně. Rozdíl v dokumentaci je však patrný ve změnových požadavcích, kde Etnetera má suverénně nejvyšší pracnost. Důvodem je nejen rozsáhlý zásah dvou změnových požadavků do hotové aplikace, ale právě i dokumentace. Ačkoli je Etnetera podobná SI1/2 co do objemu, liší se ve formě dokumentace, kde Etnetera dokumentuje podrobně z jednoho pohledu, nýbrž SI1/2 abstraktněji z více pohledů. Zanesení změn do hloubky je logicky náročnější.

V tomto specifickém ohledu vítězí Avast, kde zanesení změn nese minimální režii nad rámec úpravy kódu.

Efektivnější plánování zdrojů vývojářů během implementace u metodik Etnetera a SI1/2 by sice vedlo ke zvýšení nákladů, ale zkrátila by se doba trvání projektu. Kontinuální vytížení je navíc při průběhu reálného projektu lepší a pravděpodobnější scénář. Nadto stále zůstává prostor pro využití juniorních vývojářů během zmíněného období plánování projektu. Navíc objem pracnosti neobsahuje nezanesené paralelní aktivity. Z těchto důvodu nenahlížím v tomto měřítku na výsledky Avastu jako na negativní, neb započetím souběžných aktivit a přeplánováním zdrojů by se zbylé metodiky podobným číslům minimálně přiblížily.

Tabulka 4.6: Vyhodnocení nákladů

Pořadí	1.	2.	3.
Metodika	SI1/2	Etnetera	Avast

V posledním kritériu udávám pořadí dle naměřené pracnosti, protože teoretické uvažování přeplánování zdrojů a paralelních procesů je neexaktní přístup.

4.1.4 Srovnání

Všechny tři metodiky se ukázaly být jako funkční a každá vynikala v jiném kritériu. Relativním vítězem se stává Avast, neboť nasbíral nejvíce pomyslných prvních míst. Následuje SI1/2, která vyšla nejlépe v nákladech. Etnetera je sice na třetím místě, ale nevedla si vyloženě špatně, což dokazují dvě druhá místa.

Tabulka 4.7: Vyhodnocení metodik dle srovnávacích kritérií

Metodika	1.	2.	3.
Etnetera		doba, náklady	cíl
SI1/2	náklady	cíl	doba
Avast	cíl, doba		náklady

Určit nejlepší metodiku absolutně nelze, protože jak práce ukázala, i použití ve firmách je hodně ovlivněno faktory jako je zaměření firmy, velikost nebo potřeby zákazníka. Roli na výsledku hraje do určité míry i průběh simulace, kde odlišné nastavení parametrů ukázkového projektu by hrálo svoji roli.

4.2 Poznatky výzkumu

Jak jsem detailně rozepsal v závěru výzkumu 2.5.10, použité metodiky se ukázaly být různorodé. Nenarazil jsem na společnost, která by učebnicově

dodržovala určitou pojmenovanou metodiku. Společnosti používají metodiky odvozené, případně kombinované a doplněné o poznatky z vlastní zkušenosti.

Čím byla společnost větší, tím více byla metodika rigorózněji definována a bylo snazší identifikovat, ke kterému ze tří základních přístupů se klaní. Rovněž větší společnosti dávali rovnoměrnější důraz na analýzu, implementaci a testování, kdežto menší firmy upřednostňovaly činnost implementace.

Dále se ukázalo, že agilnímu vývoji sedí v první řadě produktový vývoj, protože společnost není omezena externím zákazníkem, a v druhé řadě menší velikost firmy, kde méně lidí si může dovolit fungovat dynamičtěji. U středně velkých zakázkových dodavatelů převládá iterativní styl vývoje.

4.2.1 Přínosy jednotlivých metodik

Ačkoli každá metodika je specifická, používá ji jiná firma a v odlišném kontextu, zmíním se krátce u každé metodiky o aspektech, které mě zaujaly a přijdou mi přínosné nehledě na okolnosti použití. Osobně doporučuji si z každého přístupu vzít to dobré.

- *SI1/2.* Tvorba dokumentace standardizovanou notací UML dává člověku znalost, která se neztratí a je přínosná hlavně při participaci na projektech s významnými klienty. Druhou výhodou je vývoj funkčního prototypu v brzké fázi projektu.
- *Senman.* Ruční testování neodborníky dodává činnosti na autentičnosti, neb dokáže věrněji simulovat chování koncového uživatele. Dekompozice projektu na dílčí podprojekty umožňuje lepší spolupráci geograficky oddělených vývojářů, potažmo projektových týmů.
- *Ackee.* Z hlediska zvýšení zastupitelnosti mě zaujala role externího dohledu osobou, jenž se na projektu nepodílí přímo, ale sleduje dění zpovzdálí a kritickýma očima. Další benefit vidím v otevřenému přístupu k novým technologiím, za účelem zpohodlnění procesu ve formě moderní infrastruktury.
- *Riganti.* Zaujal mě proces přezkoumání zadání, během kterého se aktivně hledají navzájem si odporující požadavky a šedé zóny. Dále analýza zákazníkova finančního zdraví, jež pomáhá chránit dodavatele, před budoucím překvapením při fakturaci. Třetím bodem je povinnost vyčlenění PO za stranu zákazníka, čímž se snižuje komunikační prodleva na možné minimum. V poslední řadě, standard dvou a více monitorů je maličkost, kterou každý vývojář ocení.
- *Etnetera.* Mezi některé důvody až zarážející vysoké úrovně firemní kultury společnosti takové velikosti identifikuji maticovou organizační strukturu a volnou ruku PM v naplnění jednotlivých milníků projektu (14 dohod). Oceňuji sdílení know-how ve firemní projektové akademii. Vyme-

zení přesného rozsahu testování během analýzy pokládám rovněž za přínosné. Nenápadný, ale o to podstatnější benefit vidím v začlenění pracovníka provozu už v dřívějších fázích projektu.

- *2N.* Přizpůsobení issue tracking systému potřebám firemního flow pokládám za důležitou výhodu zefektivňující práci na dennodenní bázi. Další benefit vidím v developer specification, kde vývojář po ukončení úkolu vyplní pro QA, jak úkol řešil. Obecně délka fází implementace a testování v poměru 2:1 vede k zamyšlení.

- *Adastra.* Metodika má z mého pohledu optimální míru dokumentace. Tvořená dokumentace má své opodstatnění a vhodnou hloubkou abstrakce. Využití seniorních konzultantů především v rámci analýzy dovoluje překrytí jednotlivých iterací. Zlepšení flow projektu napomáhá i umístění projektového týmu u zákazníka a nasmlouvání polštáře na změnové požadavky nad rámec původního zadání. V neposlední řadě mě zaujalo rozšíření klasifikace lidí dle zkušenosti o úroveň standard, jež vyplňuje přirozenou mezeru mezi juniorem a seniorem.

- *Avast.* Autonomie na úrovni týmů je funkční cestou při spolupráci velkého množství lidí. Dalším plusem je rozdělení zaměstnanců vertikálně dle týmů a horizontálně dle dovedností. V rámci obou skupin mají lidé meetingy, kde sdílejí informace, navzájem se vzdělávají a vylepšují proces. Používání checklistů před nasazením ukotvuje proces a zamezuje opomenutí událostí. Časté nasazování pro omezenou skupinu lepších uživatel dokáže efektivně odhalit majoritu chyb před release do celé produkce. Za zmínku stojí i soustředění veškerých firemních informací a projektové dokumentace na jednom místě – interní wiki.

4.3 Srovnání akademie a praxe

Stěžejní rozdíly mezi akademickým přístupem a praktickým řízení projektů jsem identifikoval zejména dva.

Tvorba dokumentace je aspekt, který se projevil i při simulaci, kde vlivem simulování iterativní metodiky založené na akademickém přístupu byl objem vytvořené dokumentace srovnatelný s vodopádovou metodikou z praxe. V realitě se tvoří dokumentace méně. Firmy argumentují, že dokumentují jen to, co potřebují. Osobně si myslím, že tomu tak z části není. Dokumentace je takový obětní beránek. Lidi nebaví ji tvořit, musí se udržovat a není součástí výsledné aplikace. Z těchto důvodů je dokumentace při příchodu stresových situacích jedna z prvních opomíjených věcí. Naopak v čem má praxe pravdu je to, že není nutné vytvářet stejnou dokumentaci v různé formě. Je zbytečné zachycovat diagramem a textem to, co již popisuje wireframe. Popis z více úhlů pohledu je v pořádku, pokud každý pohled přináší přidanou hodnotu. Další

rozdíl je ve formě specifikace. Praxe se hojně opírá o wireframy a grafický návrh, protože tato forma je srozumitelná pro klienta. Naopak akademické prostředí využívá dominantně popis textový spolu s UML diagramy. Tyto metody se v praxi také používají, jen v nižší míře, doplněné právě o uživatelsky přívětivější dokumentaci.

Testování je podobně opomíjeno jako dokumentace. Na povinné unit testy jsem narazil jen ve třech společnostech. Firmy mi absenci unit testů vysvětlily následovně (doplněno o můj komentář):

- *Kód je unitově netestovatelný.* Ačkoli unit testy nejsou vhodné u všech typů aplikací, tento scénář je specifický a častěji se jedná o špatný návrh rozhraní.
- *Náročná údržba, kód se často mění.* Častý refactoring je známkou absence návrhu a konceptuálního zamyšlení nad možným rozsahem projektu před začátkem implementace.
- *Není prostor a vývojářům se věří.* Krátkozraký a hazardní postoj. Je běžné, že forma a rozsah testování se s klientem domlouvá mimo činnost implementace.
- *Testování probíhá dostatečně na jiné úrovni, například smoke nebo integračními testy.* Nejvalidnější argument, nicméně je efektivnější testovat funkcionalitu přímo u zdroje než čekat na efekt obsáhlého scénáře. Testování komplexními scénáři trvá déle a je obtížné pokrýt všechny varianty vstupů. Ideální je scénáři testovat pouze vazby unitově otestovaných funkcí.

Osobně vidím příčinu v tom, že bez unit testů se dá obejít a jejich zakomponování z počátku nepřináší okamžitý benefit. Unit testy nedodají novou funkcionalitu zákazníkovi, ale dříve odhalí nefunkčnost té stávající. Dalším faktorem je nezkušenost, protože psaní unit testů vyžaduje vysoké nároky nejen na testy, ale i implementaci samotnou (první dva body argumentace).

Mimo unit testy, paleta v praxi používaných testů není tak pestrá, jako u akademického přístupu, kde se učí desítky typů testů kategorizovaných dle formy, rozsahu nebo události, při které se používají. Z pohledu formy se v praxi nejčastěji používají testy integrační a automatizované UI testy v rozsahu smoke a regresních testů. Mimo automatizovanou formu je běžně využíváno ruční testování. Automaticky se testuje omezená podmnožina funkcionality. Patří sem základní funkčnost, dále funkčnost náchylná na poruchu nebo funkčnost vyžadující zvláštní obezřetnost. Z hlediska času se testuje po dokončení jednotlivého úkolu vývojářem, automaticky s nasazením na prostředí a před nasazením do produkce (interní testování a UAT). V rámci UAT se hojně využívá zátěžové a penetrační testování. Někde, ale ne všude, jsem narazil na inkorporaci průběžného testování, kde je předpokladem dostatečná kapacita QA týmu.

4.4 Obecná doporučení

V závěru chci vyzdvihnout společné principy, na které jsem narazil u drtivé většiny metodik. Tato doporučení nemohou být pokládána za zaručený recept na úspěch, nicméně jejich inkorporace do řízení projektů nesporně má na průběh projektu pozitivní vliv. Jinak řečeno, jedná se o průnik metodik, takzvané *the best of*, jenž je základem dnešního přístupu k řízení softwarového vývoje.

1. Inkrementální vývoj
2. Využití issue tracking systému
3. DSM a vizualizace úkolů pomocí board
4. Testovací prostředí s automatickým buildem a spuštěním smoke testů
5. Wireframes / grafika jako stěžejní část dokumentace
6. Osobní schůzky a hovory se zákazníkem na minimálně týdenní bázi
7. Sdílená tvorba draft verzí dokumentů během specifikace a analýzy

Závěr

Práce ukázala, že v praxi nepřevládá jeden přístup řízení projektů, ale že používané metodiky jsou odlišné. Každá společnost má zažitou specifickou metodiku, jejíž parametry upravuje na základě empirie. Rovněž jsem narazil na to, že společnosti neřídí projekty jednou metodikou, ale volí z více metodik dle okolností projektu jako jeho typu, preferencí zákazníka nebo zkušeností projektového manažera. Ve studii jsem se proto zaměřil na metodiku nejčastěji používanou v kontextu zakázkového nebo produktového vývoje.

Zjistil jsem, že všeobecně existuje trend inkrementálního vývoje v nějaké podobě a že praktiky agilního vývoje se stávají realitou na dennodenní bázi, byť metodika se z vnějšího pohledu tváří i jako iterativní nebo vodopádová. Prostředí podporující agilitu je produktový vývoj nebo malá velikost firmy, naopak agilita zatím naráží na své limity u velkého zakázkového vývoje a softwarového vývoje navázaného na hardware. Dalším zjištěním je, že menší firmy mají tendenci být čistě implementačně orientovány, kdežto firmy větší kladou rovnoměrnější důraz mezi plánování, konstrukci a zajištění kvality.

Většina firem, zejména menších, si bere z každého ze základních přístupů (vodopád, iterativní a agilní vývoj) něco a tři ze sedmi firem jsem kategorizoval jako firmy používající přístup smíšený. Jmenovitě Senman, Riganti a 2N. Překvapením je, že jsem narazil i na praktické využití vodopádu, a to u společnosti Etnetera. Ryzím představitelem iterativní metodiky se ukázala být Adastra a nejčistší agilitou disponovaly Ackee a Avast.

Díky kvalitě získaných dat spolu s provedenou simulací na ukázkovém projektu nabízím v práci detailní zhodnocení metodik z různých hledisek. Nadto jsem vyhodnotil průnik metodik, na který nahlížím jako esenci moderního řízení softwarového vývoje. Mezi společné prvky patří zmíněný inkrementální vývoj, jehož důsledkem je více release během vývoje. Dále začlenění funkčních agilních principů jako využití board nebo DSM nehledě na obecný konstrukt metodiky. Neméně důležitá je distribuce na více vývojových prostředí zastřešených automatickými buildy a testováním v rámci continuous integration. Ohledně specifikace a plánování se klade důraz na společnou tvorbu zadání a

využití grafické představy UI části aplikace před odbornou, technickou dokumentací. Na závěr, i přes sebelepší plánování, je důležité být se zákazníkem v pravidelném kontaktu alespoň jednou za týden nehledě na etapu projektu.

Studie naplnila má očekávání, protože jsem zjistil nejen, jaké metodiky firmy používají, ale i kontext a motivaci jejich použití. Bylo tedy odpovězeno na základní otázky co, kdy a proč. Navíc zahrnutí rozmanitých firem přineslo různorodé poznatky s ohledem na kontext a práce tedy minimalizovala riziko vyvození chybného závěru pramenícího z monotónního výzkumného vzorku. Benefitem pro čtenáře studenty je nejen utvoření praktické představy teoreticky vyučovaných principů ve škole, ale i poukázání na vzájemné rozdíly a vyjasnění fungování projektů mimo sféru akademickou. Tento fakt podporuje obohacení výzkumného vzorku o metodiku vyučovanou během povinného kurzu bakalářského studia na FIT ČVUT, zjednodušený Unified Process.

Na této studii vyzdvihuji unikátní přístup k problematice použitím kvalitativních výzkumných metod, protože běžné práce na podobné téma sbírají data kvantitativně a vyhodnocení je tedy povrchní. Propracovaný výzkum dává práci jasný konstrukt. Přidanou hodnotou v mém případě je konkrétní nastavení jednotlivých metodik bez schování se za abstraktní označení. K tomu jsem zjistil motivaci a kontext použití jednotlivých metodik, což hraje zásadní roli při jejich zhodnocení. Navíc dvoustupňový proces výzkumu zaručil, že získaná data nerepresentují ideu, jak to má fungovat, nýbrž realitu jak to opravdu funguje.

Pokud bych práci dělal znovu, omezil bych se pouze na část výzkumu s případným rozšířením vzorku firem, či prodloužením pozorování po delší časový úsek pro zajištění ještě vyšší autentičnosti. Pro mě osobně je výstup z výzkumu to, co si z práce odnáším nejvíce. Provedení simulace přidává studii další úhel pohledu, nicméně její realizace je velmi obtížná. S ohledem na mé znalosti jsem simulaci provedl na hranici svých možností, jakožto absolventa bakalářského studia. Až při jejím provádění jsem si ale uvědomil řadu proměnných, které mohou hrát při průběhu projektu roli. Příkladem je zmíněný vliv souběžných projektů. Tento a další aspekty dělají ze simulace téma, které by obstálo jako samostatná závěrečná práce. Tím, že jsem do práce zařadil jak výzkum, tak simulaci, jsem vytvořil práci nad rámec rozsahu práce bakalářské. Na studii tedy nahlížím jako na spojení dvou prací.

Následná práce diplomová by mohla sloužit buď jako opětovný výzkum, který by přinesl srovnání stavu před a po, nebo jako provedení komplexní simulace rozšířenou o nabyté znalosti z magisterského studia. Výsledkem by bylo, že práce jak bakalářská, tak diplomová by se mohla detailněji zaměřit na stejnou problematiku ze své perspektivy.

Do budoucna by bylo zajímavé sledovat, jak se trend řízení projektů posouvá. S rostoucí zkušeností a rozvojem technologií lze očekávat, že agilní trend řízení projektů bude stále výraznější. Tato práce ukazuje alternativní přístup k problematice prioritizující opravdové důvody před statistickými čísly. Proto věřím, že studie může být vodítkem, potažmo základem, pro práce další.

Literatura

[1] Kruchten, P.: *The Rational Unified Process: An Introduction (3rd Edition)*. Addison-Wesley Professional, 2003, ISBN 0321197704.

[2] Bryman, A.: *Social Research Methods, 4th Edition*. Oxford University Press, 2012, ISBN 0199588058.

[3] CzechInvest: Definice malého a středního podnikatele. 2016, [cit. 2016-12-04]. Dostupné z: `http://www.czechinvest.org/definice-msp`

[4] Pressman, R.: *Software Engineering: A Practitioner's Approach*. McGraw-Hill Education, 2009, ISBN 0073375977.

[5] Sommerville, I.: *Software Engineering*. Prentice Hall, 2010, ISBN 0137053460.

[6] Ryba, P.: Rozhovor s projektovým manažerem 2N TELEKOMUNIKACE a.s. zasedací místnost Madagaskar, Modřanská 621/72, 143 00 Praha 12-Modřany, 2016-12-09.

[7] Hrach, P.: *Všeobecná encyklopedie ve čtyřech svazcích, díl 3 - M/R*. Nakladatelský dům OP, 1997, ISBN 8085841355.

[8] Krátký, T.: 1. přednáška kurzu Softwarové inženýrství II. FIT ČVUT v Praze, 2016-10-04.

[9] Jacobson, I.: A Resounding "Yes" to Agile Processes – But Also to More. *Cutter IT Journal*, 2002, ISSN 15227383.

[10] Beck, K.; aj.: Manifest Agilního vývoje software. 2001, [cit. 2017-02-18]. Dostupné z: `http://agilemanifesto.org/iso/cs/manifesto.html`

[11] Pacovský, J.: Rozhovor se senior managing consultantem ADASTRA GROUP SE. Karolinská 654/2, 186 00 Praha 8-Karlín, 2017-02-06.

[12] Lindvall, M.; Muthig, D.; Dagnino, A.; aj.: Agile software development in large organizations. *Computer*, ročník 37, č. 12, dec 2004: s. 26–34. Dostupné z: 10.1109/mc.2004.231

[13] Ambler, S.: IT Project Success Rates Survey. 2013, [cit. 2016-12-04]. Dostupné z: http://www.ambysoft.com/surveys/success2013.html

[14] Alliance, S.: The Scrum Guide. [cit. 2017-02-26]. Dostupné z: https://www.scrumalliance.org/why-scrum/scrum-guide

[15] Beedle, M.: SCRUM: An extension pattern language for hyperproductive software development. 1999, [cit. 2017-02-26]. Dostupné z: http://jeffsutherland.org/scrum/scrum_plop.pdf

[16] Haughey, D.: Understanding the project management triple constraint. 2011, [cit. 2017-02-28]. Dostupné z: https://www.projectsmart.co.uk/understanding-the-project-management-triple-constraint.php

[17] Blažek, V.; Hrabal, O.: Rozhovor s projektovými manažery Quanti s.r.o. Nikoly Tesly 1096/12, 160 00 Praha 6-Dejvice, 2017-02-28.

[18] Anon: Module 9 : Introduction to Research. 11 2009, [cit. 2016-10-29]. Dostupné z: https://www2.le.ac.uk/projects/oer/oers/lill/oers/fdmvco/module9/module9cg.pdf

[19] Hrabal, O.: Zkušební rozhovor s projektovým manažerem Quanti s.r.o. Meeting Room A, Nikoly Tesly 1096/12, 160 00 Praha 6-Dejvice, 2016-11-21.

[20] Burger, J. M.: The Foot-in-the-Door Compliance Procedure: A Multiple-Process Analysis and Review. *Personality and Social Psychology Review*, 1999. Dostupné z: 10.1207/s15327957pspr0304_2

[21] Kučera, Z.: E-mailová korespondence s advokátem Zdeňkem Kučerou. zdenek-kucera@email.cz, 2016-10-31.

[22] CzechInvest: O CzechInvestu. 2016, [cit. 2017-01-29]. Dostupné z: http://www.czechinvest.org/o-czechinvestu

[23] ČVUT; FIT: Informace k předmětu BI-SI1.2 na portálu EDUX. [cit. 2017-03-03]. Dostupné z: https://edux.fit.cvut.cz/courses/BI-SI1/

[24] ČVUT; FIT: Anotace předmětu BI-SI1.2 v systému KOS. https://www.kos.cvut.cz, [cit. 2017-03-03].

[25] Profinit: Popis předmětu BI-SI2.2 na stránkách Profinit. [cit. 2017-03-03]. Dostupné z: https://profinit.eu/univerzity/predmet/softwarove-inzenyrstvi-2/

[26] Krátký, T.: 1. přednáška předmětu BI-SI2.2 v ZS 2016/17, téma Softwarový proces. [cit. 2016-10-04]. Dostupné z: `https://profinit.eu/univerzity/predmet/softwarove-inzenyrstvi-2/`

[27] Černý, D.: Rozhovor s majitelem Senman s.r.o. kantýna, Argentinská 286/38, 170 00 Praha 7-Holešovice, 2016-11-25.

[28] Senman: Pozorování v Senman s.r.o. Argentinská 286/38, 170 00 Praha 7-Holešovice, 2017-02-10.

[29] Veselý, D.: Rozhovor se spolumajitelem Ackee, s.r.o. Karolinská 650/1, 186 00 Praha 8-Karlín, 2016-12-02.

[30] Herceg, T.: Rozhovor s majitelem RIGANTI s.r.o. Sokolovská 352/215, 190 00 Praha 9-Vysočany, 2016-11-29.

[31] Riganti: Pozorování v RIGANTI s.r.o. Sokolovská 352/215, 190 00 Praha 9-Vysočany, 2017-02-02.

[32] Randa, J.; Beran, V.: Rozhovor s team leaderem a projektovým manažerem Etnetera a.s. Jankovcova 1037/49, 170 00 Praha 7-Holešovice, 2017-01-13.

[33] Etnetera: Pozorování v Etnetera a.s. Jankovcova 1037/49, 170 00 Praha 7-Holešovice, 2017-02-14.

[34] Turek, J.: Rozhovor se senior projekt manažerem během pozorování v ADASTRA GROUP SE. Karolinská 654/2, 186 00 Praha 8-Karlín, 2017-02-09.

[35] Kozelka, D.: Rozhovor s Division Director ADASTRA GROUP SE. zasedací místnost Vltava, Karolinská 654/2, 186 00 Praha 8-Karlín, 2017-01-11.

[36] Adastra: Pozorování v ADASTRA GROUP SE. Karolinská 654/2, 186 00 Praha 8-Karlín, 2017-02-09.

[37] Honzák, L.: Rozhovor s Online Operations Director Avast Software s.r.o. Enterprise Office Center, Pikrtova 1737/1a, 140 00 Praha 4, 2016-12-01.

[38] Avast: Pozorování v Avast Software s.r.o. Enterprise Office Center, Pikrtova 1737/1a, 140 00 Praha 4, 2017-02-16.

[39] Zima, M.: Rozhovor se senior produkt manažerem během pozorování v Avast Software s.r.o. Enterprise Office Center, Pikrtova 1737/1a, 140 00 Praha 4.

[40] Budáč, M.: Rozhovor se senior projekt manažerem během pozorování v Avast Software s.r.o. Enterprise Office Center, Pikrtova 1737/1a, 140 00 Praha 4.

Příloha **A**

Seznam použitých zkratek

AM Account manažer

API Application programming interface

BA Business analytik

BE Back end

BPMN Business Process Model and Notation

CASE Computer-aided software engineering

CI Continuous integration

CMS Content management system

CR Codereview

DSM Daily standup meeting

ERP Enterprise resource planning

FE Front end

FL First line

FP Funkční požadavek

IDE Integrated development environment

IS Informační systém

IT Informační technologie

MD Man day

MTP Master test plan

PdM Produktový manažer

PM Projektový manažer

PO Product owner

QA Quality assurance

SI Softwarové inženýrství

SLA Service level agreement

TL Team leader

UML Unified Modeling Language

UI User interface

UP Unified Process

UX User experience

Příloha B

Doplňující výzkumné materiály

B.1 Tabulka potenciálních společnosti

Společnost	Kategorie	Blízký kontakt	Interview	Pozorování
Ackee	malá	Michaela Trnková	2. 12. 2016	
Aira	malá	Jan Rýšavý		
Intens	malá	Ondřej Hrabal		
Senman	malá	Marek Polčák	25. 11. 2016	10. 2. 2017
Farmis	malá	Marek Polčák		
Dr. Max (IT oddělení)	malá	Marek Polčák		
HaugLand	malá	Marek Polčák		
CEOS data	malá	Marek Polčák		
Ataccama	střední	Michaela Trnková		
Riganti	střední	Ladislav Šesták	29. 11. 2016	2. 2. 2017
2N telekomunikace	střední	Marek Polčák	9. 12. 2016	
Digital Engines	střední	Marek Polčák		
TollNet	střední	Jiří Novák		
Etnetera	střední	Samuel Butta	13. 1. 2017	14. 2. 2017
Seznam	velká	Václav Podlipný		
Adastra	velká	Michal Kureš	11. 1. 2017	9. 2. 2017
Alza	velká	Eugen Protopapas		
Microsoft	velká	Ladislav Šesták		
IBM	velká	Stanislav Mikeš		
Avast	velká	Václav Podlipný	1. 12. 2016	16. 2. 2017
Profinit	velká	Tomáš Krátký		
Unicorn	velká	Tomáš Krátký		

B.2 Etnetera 14 dohod

- *Zadání* Každý projekt má zadání. Není důležitá forma, ani systém ve kterém je uloženo. Důležité je nalinkovat odkaz na zadání v BANGu[4].

- *Odhad* Každý projekt má odhad pracnosti, popř. dalších dodávek, SLA či provozních nákladů. Odhad vnímáme jako rozpad pracovních balíků a úkolů (WBS), doplněný o předpoklady a rizika. Odhad je určitá forma předimplementační analýzy.

- *BA* Každý projekt má interní kalkulaci. V případě některých projektů je kalkulace součástí odhadů, u většiny se však jedná o formát BA. Je důležité, aby interní kalkulaci validoval podpisem někdo další. Zajistíme tak, že kalkulace bude obsahovat všechny důležité části a bude ve firmě povědomí o nutnosti budoucí alokace lidí na projekt.

- *GO* Pro start prací na projektu potřebuje PM jednoznačné potvrzení od AM (nejlépe do mailu). V případě, že klient ještě nepodepsal objednávku, dává garanci, že můžeme začít pracovat AM.

- *Kickoff* Doporučujeme dělat kickoff schůzky u každého projektu (interní vždy, po zvážení také kickoff s klientem).

- *Komunikační plán* Je doporučeno připravit si pro projekt komunikační plán pro účely řízení zainteresovaných stran.

- *Výkazy* Každý projekt je vykazován za všechny billable role (i PM).

- *Rizika.* PM je povinnen celý projekt pracovat na snižování jeho nejistoty, tedy samozřejmě i možných rizik.

- *Dohoda s DPL.* Jelikož je následný provoz, SLA a support podstatnou složkou projektů, je třeba se vždy zamyslet, zda do projektu potřebuji zapojit Deployment (práce, provoz, HW, SLA, maintain ...).

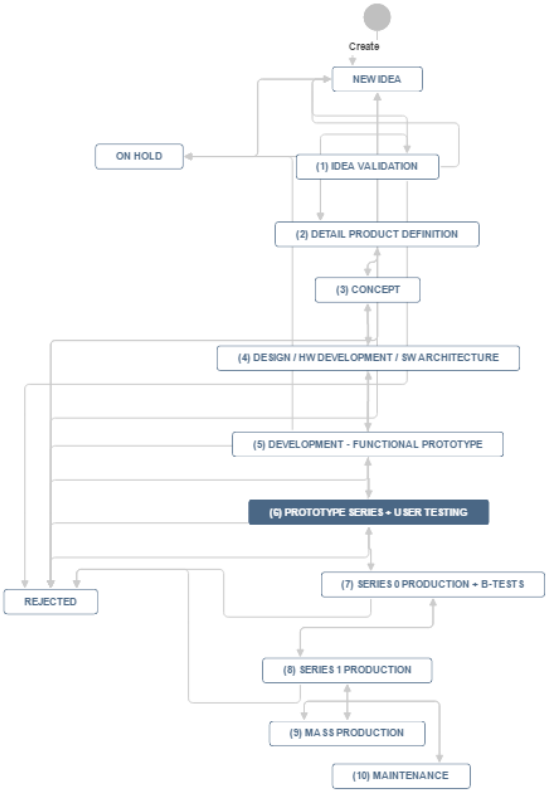
- *QA* Je nutné se u každého projektu dohodnout s QA, jaké typy testů (uživatelské, zátěžové, penetrační) a v jakém rozsahu bude třeba vykonat (odhady pracnosti).

- *Spouštěcí plán.* Pokud nejde o rutinní spouštění nové verze stávajícího webu, je velice doporučeno připravit spouštěcí plán projektu. Velké, důležité weby a weby se zapojením dalších stran do spouštěcího procesu by neměly být bez spouštěcího plánu vůbec pouštěny.

[4]BANG je vlastní IS Etnetery. Slouží pro evidenci projektů a výkazy práce.

- *Retrospektiva.* Je doporučeno ukončovat projekty retrospektivou týmu, probrat co se povedlo a co nikoliv -> poučení do budoucna. Retrospektivu organizuje PM. Čím větší, důležitější, či pokaženější projektu, tím spíše by měla být retrospektiva provedena.
- *Akceptace.* Každý projekt je ukončen předávacím protokolem, který opravňuje k fakturaci. Je nutné zajistit, aby protokol obsahoval správný kód projektu a také, aby ho akceptovala osoba, která je k tomu ze strany zákazníka pověřena.
- *Předání do supportu.* Projekty, které nezůstávají v klientském týmu jsou předávány do oddělení Evolve. Koncového First line pracovníka je třeba do projektu zapojit co nejdříve, aby byl přechod do běžného supportu co nejrychlejší a PM se mohl věnovat dalším projektům.

B.3 2N projektové flow



Obrázek B.1: 2N projektové flow

Příloha C

Obsah přiloženého CD

```
readme.txt ..... stručný popis obsahu CD
thesis ..... text práce
    thesis.tex ..... zdrojová forma práce ve formátu LaTeX
    thesis.pdf ..... text práce ve formátu PDF
research ..... výzkumné přílohy
project ..... podklady simulace
```