Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Vlastnosti moderních metodik vývoje software

Plzeň, 2019 David Langmaier

PROHLÁŠENÍ

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 14. května 2019 . . . . . .

David Langmaier

ABSTRAKT

Tato práce obsahuje popis moderních frameworků pro rozsáhlé projekty. Konkrétně se jedná o frameworky Disciplined Agile Delivery (DAD), Large-Scale Scrum (LeSS) a Scaled Agile Framework (SAFe). Podrobně popsány jsou jejich základní charakteristiky, klíčové hodnoty, role, artefakty a činnosti. Na základě zjištěných poznatků je definována sada praktik, které jsou společné pro všechny tyto frameworky nebo jsou klíčovou součástí agilních metodik. Tyto praktiky jsou popsány tak, aby mohly tvořit základ pro detekci experimentálním nástrojem SPADe, jenž slouží pro zachycení dat o projektech a umožňuje jejich následnou analýzu. V práci jsou zachyceny také výsledky detekce na sadě studentských projektů.

KLÍČOVÁ SLOVA

Disciplined Agile Delivery, DAD, Large-Scale Scrum, LeSS, Scaled Agile Framework, SAFe, vývoj software, agilní metodiky, rozsáhlé projekty, SPADe

ABSTRACT

This thesis contains a description of modern frameworks for projects at scale. Specifically, it includes Disciplined Agile Delivery (DAD), Large-Scale Scrum (LeSS) and Scaled Agile Framework (SAFe). Their main characteristics, key values, roles, artifacts, and activities are described in detail. Based on the findings, a set of crucial practices was defined. These practices are common for the frameworks or are a key part of agile methods in general. They are described in such a way so that they can be used as a basis for detection by an experimental tool called SPADe. This tool is used to capture project data and allows their further analysis. Results of the detection on a set of student projects are reported in the thesis.

KEYWORDS

Disciplined Agile Delivery, DAD, Large-Scale Scrum, LeSS, Scaled Agile Framework, SAFe, software development, agile methods, scaled projects, SPADe

LANGMAIER, David *Vlastnosti moderních metodik vývoje software*: diplomová práce. Plzeň: Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky, 2019. 83 s. Vedoucí práce byl Doc. Ing. Přemysl Brada, MSc., Ph.D.

PODĚKOVÁNÍ

Velice děkuji vedoucímu diplomové práce panu Doc. Ing. Přemyslu Bradovi, MSc., Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci. Dále bych chtěl poděkovat také panu Ing. Petru Píchovi za konzultace související s nástrojem SPADe.

OBSAH

1 ÚVOD

Cílem této práce je průzkum moderních metodik a technik pro vývoj software, které byly navrženy v posledním desetiletí, případně je jejich vývoj stále v aktivním procesu. Na základě tohoto průzkumu následně dojde k výběru podmnožiny několika metodik, které budou posléze popsány podrobněji. Důraz bude kladen především na klíčové hodnoty a stěžejní principy zvolených metodik. Budou také popisovány jejich základní role, artefakty a činnosti.

Na základě zjištěných informací bude vybrána sada praktik, které jsou pro moderní metodiky charakteristické a důležité. Tyto praktiky budou podrobně popsány a dojde také k definování jejich základních symptomů. Na základě těchto symptomů pak bude možné praktiky detekovat pomocí experimentálního nástroje SPADe, který umožňuje zachycení dat a analýzu projektů softwarového vývoje. Popis tohoto nástroje a jeho doménového modelu bude taktéž součástí práce.

Funkčnost vybraných popisovaných praktik bude ověřena na testovací sadě dat z příkladových projektů. Dojde k vyhodnocení zjištěných informací a k jejich porovnání s reálnými daty projektů. Budou také popsány možné problémy a nedostatky v rámci samotné detekce i v souvislosti s nástrojem SPADe.

2 ITERATIVNÍ VÝVOJ A AGILNÍ METODIKY

Projektové řízení v oblasti softwarového vývoje je velmi specifická činnost. Je zde vskutku velké množství různých proměnných, které je potřeba brát v potaz. Oproti některým jiným oblastem (např. řízení v oblasti výroby) je vývoj software mnohem méně předvídatelnou činností. Proto je potřeba brát tuto skutečnost na vědomí a již od úplného začátku s ní počítat.

Projektoví manažeři se velmi často potýkají s následujícími problémy:

- Zřídkakdy je možné dokončit detailní, a v budoucnosti již neměnnou, specifikaci vyvíjeného produktu ještě před tím, než započne samotná konstrukční fáze projektu.
- Na počátku projektu je velice obtížné odhadnout celkové náklady. To se týká jak finančních nákladů, tak také časové náročnosti.
- Na počátku projektu nelze přesně stanovit podrobný projektový plán. Není tedy možné jasně identifikovat, definovat a správně seřadit všechny dílčí aktivity. Je vhodné všechny tyto činnosti průběžně přizpůsobovat aktuální situaci a vývoji projektu.
- Často je potřeba přizpůsobovat se neočekávaným skutečnostem a změnám. Četnost těchto nutných změn a odchylek v projektovém plánu je poměrně vysoká.

Z výše uvedeného je jasně patrné, že není příliš vhodné aplikovat tzv. vodopádový životní cyklus. Tento přístup je založen na tom, že je na samém počátku projektu jasně naplánován celý životní cyklus vývoje projektu – je předem naplánována podrobná specifikace, odhady apod. Tyto odhady jsou však stanoveny na základě mnoha různých faktorů, které se však zpravidla v průběhu projektu mnohdy mění. Některé faktory, které značně komplikují použití vodopádového modelu, jsou následující:

- Klient či uživatel obtížně předává své znalosti a potřeby. Může docházet i k nedorozuměním a nepochopením ze strany analytika nebo konzultanta.
- Klient/uživatel si v počátcích projektu často není zcela jist tím, co přesně chce a potřebuje (IKIWISI[1]).
- Mnoho detailů o potřebách zadavatele bývá odhaleno až v průběhu fáze vývoje. Tyto drobnosti pak mohou výrazně ovlivnit projektový plán. V některých případech může dokonce docházet ke změnám názoru klienta, tím pádem i k úpravě požadavků.
- Externí faktory – změna v legislativě, konkurenční produkt, změna preferencí externího uživatele apod.

Z výše zmíněných důvodů je zřejmé, že bylo nutné radikálně změnit přístup k projektovému řízení. Moderní přístupy se mnohem více zaměřují na to, jak se vypořádat s vy-

[1] I'll Know It When I See It – „budu to vědět, až to uvidím“. Tento akronym přesně vyjadřuje chování klienta, který se zpravidla není schopen o některých věcech rozhodovat s předstihem. Často jej některé věci napadají až ve chvíli, kdy je podstatná část funkcionality již hotová a on si ji může reálně vyzkoušet. Projektový management by si měl být této skutečnosti dobře vědom a počítat s ní.

sokou mírou nejistoty, a také se dokáží mnohem lépe přizpůsobit neočekávaným změnám v průběhu projektu. Velké množství těchto moderních metodik proto staví na základech iterativního, případně agilního, přístupu. [1]

2.1 Iterativní a inkrementální vývoj

Iterativní vývoj je založen na velmi jednoduchých principech. Hlavním z nich je rozdělení projektu na několik menších částí (tzv. iterací). Každá z těchto iterací má pak stanoveno několik základních činností (fází), které se v rámci každé iterace vždy opakují. Tyto fáze jsou zpravidla čtyři:

- analýza požadavků,
- návrh (design),
- konstrukce (implementace, programování),
- testování.

Délka jednotlivých iterací může být poměrně variabilní. Je však vhodné plánovat je na kratší časový úsek. Čím kratší je totiž každá iterace, tím snáze a přesněji ji lze naplánovat. Doporučená délka iterace je v rozmezí mezi jedním a šesti týdny. Počet iterací se samozřejmě odvíjí od jejich délky a od rozsahu daného projektu.

Dlouhé projekty, jejichž délka výrazněji přesahuje jeden rok, je taktéž poměrně obtížné odhadnout a naplánovat (stejně jako tomu je u dlouhých iterací). Proto je vhodné v takových případech rozdělit projekt na několik menších podprojektů a ke každému z nich pak přistupovat individuálně.

Velmi důležitou charakteristikou iterativních metod je také skutečnost, že na konci každé iterace by měla vzniknout stabilní verze projektu. Tato verze by samozřejmě měla být plně integrována s již vytvořenou funkcionalitou a měla by také být kompletně otestována. To umožňuje mnohem dřívější prezentaci části produktu zákazníkovi, čímž se lze snadno vyhnout budoucím komplikacím zmíněným v úvodu kapitoly. Tyto prezentace je vhodné provádět průběžně (ideálně po každé iteraci). V některé literatuře je tak název těchto iterativních metodik rozšířen ještě o přívlastek inkrementální. [1]

Mnoho zásahů od externích stakeholderů lze v projektu připustit při plánování iterace (tzn. před jejím začátkem). Ve skutečnosti jsou tyto zásahy dokonce i vítány a pozitivně hodnoceny. V průběhu iterace je však jakýkoli externí požadavek na změnu nežádoucí. Před koncem iterace je také vhodné zamyslet se nad jistou retrospektivou. Poučení z uplynulé iterace lze pak přenést do té následující nebo jej využít v budoucnosti.

2.2 Agilní vývoj

Agilní přístup je novější než ten iterativní. Dá se však říci, že spousta jeho aspektů vychází z těch iterativních. Některé z nich jsou na něm dokonce zcela založeny. Využívají se zde krátké iterace, které jsou jasně časově vymezeny. Aplikuje se zde také evoluční přístup (celkový systém je budován na základě inkrementálních přírůstků) a plánování je adaptivně prováděno v průběhu projektu. Jedním z hlavních cílů agilních metodik je rychlá a bezproblémová reakce na změnu.

Skupina odborníků v oblasti iterativních a agilních metodik dokázala shrnout jejich základní myšlenky a vznikl tak tzv. agilní manifest [2]:

- lidé a jejich vzájemná interakce jsou důležitější než procesy a nástroje,
- fungující software je důležitější než důkladná a kompletní dokumentace,
- součinnost a spolupráce se zákazníkem je důležitější než vyjednávání o smlouvě,
- pružná reakce na změnu je důležitější než striktní dodržování plánu.

Na základě agilního manifestu byly následně založeny a definovány klíčové principy a postupy agilních metodik. Jedná se např. o průběžné a časté dodávky funkčních částí software, akceptovaní změn i v průběhu projektu, společná práce a koordinace spokojených pracovníků na denní bázi či tvorba retrospektivy a následné přizpůsobení se zjištěným poznatkům.

3 PRŮZKUM MODERNÍCH METODIK A PRAKTIK

V rámci této kapitoly jsou popsány metodiky a praktiky, které byly navrženy v posledních přibližně deseti letech, případně je jejich vývoj v současné době stále aktivní. Některé z nich jsou poměrně známé (SAFe, DAD, LeSS, DevOps či DSDM), ostatní byly objeveny v rámci získávání informací o zmíněných populárnějších frameworcích.

Pro účely podrobnějšího popisu byly vybrány frameworky SAFe, DAD a LeSS. Tato volba byla učiněna především proto, že všechny tři frameworky se zabývají problematikou škálování agilního přístupu pro rozsáhlé projekty. Díky tomu lze předpokládat, že obsahují mnoho společných charakteristik, které bude možné detekovat nástrojem SPADe. Také jsou poměrně populární a existuje k nim dostatečné množství literatury i dalších informačních zdrojů. Jejich podrobný popis se nachází v kapitolách 4-6.

3.1 DevOps

DevOps (Development and Operations) je sadou praktik, které umožňují automatizaci procesu mezi softwarovým vývojem a IT oddělením, což umožňuje rychlejší a spolehlivější vývoj, testování i nasazování projektových přírůstků. Tyto praktiky jsou v posledních letech velice populární. Základním konceptem DevOps je vybudování kultury vzájemné spolupráce mezi odděleními a týmy, které v historii zpravidla fungovaly zcela nezávisle. Mezi hlavní benefity patří zvýšení důvěry mezi různými články společnosti, rychlejší dodávky software, schopnost lepšího řešení kritických problémů a snazší řízení nezbytných pracovních činností. [3] [4]

3.2 Disciplined Agile Delivery

Disciplined Agile Delivery (DAD) je framework pro rozsáhlé projekty, který je orientován na lidi, jejich vzájemnou interakci a na proces učení. Jedná se o hybridní (složen z poznatků z několika různých metodik) agilní přístup pro dodávku komplexního IT řešení. Životní cyklus dodávky software je v rámci DAD řízen minimalizací rizika i přidanou hodnotou. Jedná se o škálovatelný framework, jenž se soustředí na pokrytí potřeb celé organizace.

Důležitou charakteristikou DAD je také jeho zaměření na cíl – činnosti, které nepomáhají k dosažení žádného z cílů, nemá smysl provádět. Framework tak nabízí mnoho různých řešení, přičemž žádné striktně nepředepisuje. Pouze poskytuje množinu alternativ, u kterých následně popisuje jejich klady, zápory a možné problémy. Díky tomu si každá společnost i projekt dokáže definovat takovou strategii, která bude perfektně vyhovovat konkrétním situacím a potřebám. [5]

3.3 DSDM Agile Project Framework

Dynamic System Development Method (DSDM) je agilní metodika, která je široce známá již od devadesátých let. Původně byla založena na principech metodiky RAD (Rapid Application Development). Její vývoj však neustal, je stále aktivní a v roce 2014 byla vydána nová verze agilního frameworku pod názvem DSDM Agile Project Framework.

Tento framework obsahuje bohatou sadu rolí a zodpovědností, které jsou uzpůsobeny potřebám projektů v korporátním prostředí. Nabízí také praktiky nezbytné k udržení kontroly nad agilními projekty. Framework je neustále přizpůsobován moderním agilním trendům, přičemž je kladen speciální důraz na tvorbu pouze minimální dokumentace v inicializační fázi projektu. [6]

3.4 GROWS

Metodika GROWS je soubor praktik a přístupů, které popisují, jak efektivně integrovat vývoj software a proces učení v rámci organizace. Jedná se o empirický proces, který obsahuje krátké cykly, na jejichž konci je vždy poskytován feedback. Metodika je založena na čtyřech základních konceptech – kontinuální vývoj, rychlé experimenty (umožňují rychlé zodpovídání technických i jiných otázek), kontinuální učení (na osobní i podnikové úrovni) a začlenění všech součástí organizace do procesu vývoje software, díky kterému budou všechny články efektivně spolupracovat. [7]

3.5 Large-Scale Scrum

Large-Scale Scrum (LeSS) není žádnou novou implementací či vylepšením standardního Scrumu. Jedná se spíše o sadu rozšíření pro rozsáhlé projekty, přičemž jsou však zachovány všechny základní principy, praktiky a zásady klasického Scrumu. Je také kladen důraz na jednoduchost frameworku a na minimální míru restrikcí.

LeSS poskytuje 2 různé typy frameworku, mezi kterými je možno volit na základě velikosti projektu. První z frameworků je určen pro projekty čítající 2–8 týmů, druhý je vhodný pro projekty skládající se z více než 8 týmů. Oba tyto typy se zakládají na společných principech, druhý z nich je pouze rozšířen o některé dodatečné role, artefakty a činnosti. [8]

3.6 Nexus

Framework Nexus byl vytvořen Kenem Schwaberem (jedním ze spoluzakladatelů metodiky Scrum) a byl zveřejněn jeho organizací Scrum.org v roce 2015. Je založen primárně na Scrumu a využívá iterativní a inkrementální přístup při vývoji software. Jeho snahou je rozšířit široce známý Scrum pouze minimálně, aby zůstaly zachovány jeho hlavní myšlenky a jeho jednoduchost.

Nexus se zabývá především škálováním Scrumu. Řeší tak mnohé časté problémy, které souvisejí s rozsáhlými projekty a s jejich řízením. Tyto obtíže se snaží eliminovat pomocí minimalizace závislostí mezi jednotlivými projektovými týmy, přičemž je zachován důraz na rozhodování na nejnižší úrovni. Framework se také zabývá problémy v oblasti integrace jednotlivých částí systému. [9]

3.7 Oikosofy

Zakladatelé společnosti Oikosofy pracují jako konzultanti v oblasti agilního vývoje software. Poskytují dalším společnostem rady, jak dosáhnout dodání vysoce hodnotného produktu, který je však zároveň finančně výnosný a zákazníci jej oceňují. Také upozorňují na sadu problémů, kterým v současné době organizace čelí. Zároveň s tím také nabízejí rady, jak tyto problémy jednoduše a systematicky řešit, případně jak jim zcela předcházet.

Jejich zkušenosti daly vzniknout agilnímu frameworku Oikosofy. Jedná se o metodiku, která poskytuje sadu rad při dodržování dané projektové strategie, pomáhá budovat vnitropodnikové prostředí a eliminovat organizační problémy. Také poskytuje nástroje, které umožňují měřitelnost dodržování zvolené strategie a jejích výsledků. Framework klade speciální důkaz na neustálý proces učení v rámci organizace, ve které si všichni předávají informace a znalosti na každodenní bázi. [10]

3.8 Scaled Agile Framework

Scaled Agile Framework (SAFe) je jedním z předních frameworků pro škálování agilního přístupu v organizacích. Je navržen tak, aby společnostem pomohl kontinuálně a efektivněji generovat hodnotu v pravidelných a předvídatelných intervalech. Mimo soubor agilních praktik je framework založen také na štíhlém přístupu, přičemž jsou v jeho rámci využívány také poznatky z nejrůznějších dalších metodik. Do agilního procesu je zapojeno vedení organizace, střední management i jednotlivé týmy, díky čemuž lze dosáhnout adopce agility napříč všemi úrovněmi organizační struktury. [11]

3.9 SEMAT

SEMAT (Software Engineering Method and Theory) je iniciativa komunity lidí (okolo 2000 osob) a několika společností a univerzit po celém světě, která vznikla v roce 2009. Jejich společným cílem je definice společných základů a charakteristik pro softwarové inženýrství. To je něco, co až do této doby nikdy nebylo komplexně definováno.

SEMAT tedy popisuje proces vývoje software založený na pevné teorii, osvědčených principech a ověřených postupech. Základem je naučit týmy vizualizovat a porozumět procesu vývoje projektu a umožnit jim nejen snadné předávání znalostí, ale také jejich rychlému pochopení a osvojení. [12] [13]

4 SCALED AGILE FRAMEWORK

Scaled Agile Framework (SAFe) je volně dostupná znalostní báze, která je založena na úspěšných a prověřených praktikách. Je založena především na agilních a Lean principech vývoje software. Tento framework je vyvinut a poskytován organizací Scaled Agile, Inc. a jeho účelem je pomoci organizacím v doručení kvalitního produktu s využitím moderních přístupů a technik. Framework SAFe je vyobrazen v příloze A.

Informace pro popis tohoto frameworku byly primárně čerpány z knižní publikace, která se SAFe věnuje [14]. Mezi další zdroje, jenž byly v rámci práce použity, patří oficiální online znalostní báze [15] a sada zveřejněných případových studií [16]. Obsah této kapitoly představuje konsolidovaný stručný popis metodiky SAFe.

Případové studie ukazují, že velké množství společností, které tuto metodiku dokázaly implementovat, zaznamenalo velmi dobré výsledky. Konkrétněji lze hovořit o zlepšení v několika klíčových oblastech. Jedná se např. o pozitivní změny v následujících indikátorech:

- zrychlení dodání finálního produktu na trh (o 30–70 %),
- zaznamenání posunu v celkové produktivitě týmů (o 25–75 %),
- zlepšení kvality produktu (o 20–50 %),
- znatelně vrostla spokojenost a odhodlanost členů týmu (o 10–50 %).

V následujících částech jsou podrobně popsány klíčové hodnoty a principy frameworku. Poté následuje popis tří jeho hlavních úrovní (týmová, programová a portfolio) a spolu s nimi jsou rozebrány také definované role, artefakty a činnosti. Na závěr jsou popsány případové studie frameworku.

4.1 Klíčové hodnoty

Metodika SAFe definuje čtyři klíčové hodnoty, které považuje za zcela zásadní. Je zapotřebí, aby měl vyšší management všechny tyto zásady na paměti a respektoval je. [14]

4.1.1 Sjednocené myšlení

Je nezbytné, aby všechny zainteresované osoby v projektu měly sjednocené myšlení a hodnoty. To především zahrnuje, aby všichni směřovali ke stejnému strategickému cíli. Neměla by zde panovat nadbytečná a kontraproduktivní soutěživost, která může následně vést ke zhoršení celkové nálady v týmu. O všech zásadních změnách ve směřování projektu by měli být všichni náležitě informováni (zvláště pokud se jedná o distribuované týmy). Vše by mělo být maximálně transparentní a do projektu by měli být průběžně začleňováni všichni stakeholdeři.

4.1.2 Integrovaná kvalita

Kvalita je jedním z klíčových parametrů projektu a sehrává klíčovou roli při vyhodnocování úspěšnosti projektu. Je vhodné odlišovat kvalitu dodávaného řešení a kvalitu řízení projektu (celkového programu). Kvalita produktu zajišťuje spokojenost zákazníka či koncového uživatele. Kvalita programu pak umožňuje rychlejší dodání, lepší předvídatelnost, optimálnější reakci na změnové požadavky a neočekávané situace či omezení rizika. Tato kvalita musí být integrována do všech procesů a do každého přírůstku systému. Neměla by tedy být vyhodnocována a vynucována dodatečně, což zpravidla způsobuje nutnost dalších úprav, následkem čehož narůstá zpoždění.

Kvalitu je tedy nutné zahrnout do všech oblastí. Důležitá je nejen kvalita samotného vyvíjeného software, kterou lze dosáhnout např. pomocí průběžné implementace a nasazování, párového programování či refactoringu. Je třeba se soustředit i na hardware (časté návrhové cykly, průběžná integrace, práce ve vícečlenném týmu, investice do otestování infrastruktury apod.) a integraci (tvorba dema systému, průběžné testování integrace). Podstatnou část tvoří také kvalita v oblasti dodržování klientových požadavků a vyhovění jeho potřebám – k tomuto účelu je vhodný např. artefakt specifikace softwarových požadavků (tzv. SRS – *Software Requirements Specification*), který umožňuje lepší ověření klientských nároků.

4.1.3 Transparentnost

Je naprosto stěžejní, aby nedocházelo ke ztrátě informací mezi jednotlivými projektovými segmenty. Informace musejí být distribuovány průběžně a všem osobám, které z nich mohou profitovat. Je tedy nutné budovat a udržovat důvěru. V opačném případě se informace nešíří správně, dochází k jejich ztrátám (ať už neúmyslným či zcela záměrným) a projekt přestává být řiditelný – nelze totiž spolehlivě řídit něco o čem zodpovědní lidé nemají kompletní informace.

Všechny kompetentní osoby musejí mít přístup k projektovým backlogům – ať už týmovým či produktovým. Také je třeba zajistit, aby všichni rozuměli cílům v rámci prací na aktuálním přírůstku systému. Vhodné je také sdílení Kanbanových tabulí, což umožňuje lepší vizualizaci čekajících, aktuálně prováděných či dokončených úkolů. Pokud je vše skutečně transparentní, není složité správně vyhodnocovat aktuální průběh projektu.

4.1.4 Dodržování programu

Touto poslední klíčovou hodnotou je myšleno dodržování všech základních principů a praktik metodiky SAFe. Je nebytné, aby tyto postupy dodržovaly všechny úrovně společnosti. Vrcholový management je musí vyžadovat, střední management plně chápat, prosazovat

a vysvětlovat a ostatní pracovníci by je měli přijmout a dodržovat. Veškeré změny musejí být provedeny globálně, komplexně a všichni s nimi musejí být důkladně srozuměni.

4.2 Základní principy

V průběhu projektu často dochází k nejrůznějším problémům a komplikovaným situacím. Tyto události často zapříčiňují to, že se soustředění managementu i ostatních členů týmu odkloní od klíčových zásad – často zcela nevědomky. Zpravidla pak dochází pouze k lokálnímu řešení nastalého problému. Takováto řešení však nemusejí platit obecně a mohou tvořit další obtížné situace v jiných oblastech.

Z tohoto důvodu SAFe doporučuje soustředit se na několik zásadních principů, které jsou velmi obecné, stabilní a mnohokrát prověřené. Tyto praktiky jsou založeny na devíti základních konceptech, které vzešly především agilních a Lean metodik, systémového myšlení a z mnoha pozorování a vyhodnocování úspěšných společností. [14]

4.2.1 Vnímání ekonomické perspektivy

Ekonomickou perspektivou zde není myšlena pouze orientace na finanční aspekty. Jedná se zde také o přemýšlení o hospodárném časovém rozvržení, alokaci a rozložení lidských zdrojů (pracovní síly), a také o funkčním obsahu produktu. Často je totiž velmi náročné rozhodnout, které funkcionality opravdu generují tu nejvyšší přidanou hodnotu pro zákazníka či koncového klienta.

V rámci těchto otázek by se měli umět rozhodovat všichni klíčoví členové týmu (nikoli pouze management). To však bývá problém, jelikož každý člověk takové schopnosti buďto nemá, nebo se na daný problém soustředí s takovou mírou detailu, že zcela ztrácí jakýkoli nadhled. Je tedy obvyklé, že se takováto rozhodnutí eskalují výše, čímž dochází k jisté centralizaci. Tento způsob pak ale vytváří zbytečné prostoje, což není žádoucí.

Jedním z nástrojů, které maximalizují ekonomickou hodnotu pro zákazníka, jsou brzké a frekventované dodávky částí produktu. Zákazník jej tak může používat mnohem dříve, což může poskytovat konkurenční výhodu. Je tak také generována vyšší kumulativní hodnota než u kompletního produkt dodaného mnohem později. Zákazníkův feedback je obdržen dříve a častěji, díky čemuž se lze mnohem lépe ekonomicky rozhodovat i decentralizovaně. Lze tak přenést část zodpovědnosti i na klienta (nebo se alespoň inspirovat jeho postřehy), jelikož on sám dokáže nejlépe určit, co mu přinese největší ekonomický užitek.

SAFe také definuje 5 základních parametrů, na které je při vývoji SW třeba dbát:

- náklady na vývoj,
- doba vývoje,

- náklady spojené s nasazením a údržbou produktu,
- ekonomická hodnota pro zákazníka (užitečnost),
- risk.

Všechny tyto aspekty je třeba brát v potaz a nalézt mezi nimi rovnováhu tak, aby celková ekonomická hodnota byla co nejvyšší. Poslední zmíněný parametr, tedy risk, je specifický tím, že se dá jen velmi obtížně vyčíslit a ovlivnit (často také vůbec není přímo spjat s ostatními parametry).

4.2.2 Aplikace systémového myšlení

Aplikace systémového myšlení je jednou z nejdůležitějších zásad metodiky SAFe. Je nutné, aby na celé řešení bylo nahlíženo jako na jeden velký systém, ve kterém jsou všechny součásti velmi úzce propojeny. Kromě projektových procesů by mělo docházet také k optimalizaci celého prostředí, ve kterém projekt vzniká – neboť i to je součástí systému. [14]

Při vývoji aplikace je nutné se na ni zaměřit jako na celek. Její jednotlivé komponenty musí být v rámci architektury dobře navrženy a musí spolu vzájemně správně komunikovat. Vylepšení jedné z nich nemusí nezbytně znamenat, že se zvýší i celková hodnota či výkon celé aplikace.

I celá organizace a její struktura jsou systém – lidé, procesy, řízení. Dobrá kultura v rámci společnosti musí být udržována schopnými leadery. Lidé by zde měli bez problémů spolupracovat a komunikovat. Ke klientům i dodavatelům musí být přistupováno jako k rovnocenným partnerům (a budovat s nimi korektní a dlouhotrvající vztahy). Opět je zde třeba aplikovat systémové principy – zaměřit se a optimalizovat organizaci jako celek, nikoli pouze její části.

Obdobné principy platí taktéž pro optimalizaci všech procesů a postupů, které generují výslednou hodnotu produktu. Je nutné zaměřit se na jednotlivé činnosti v rámci procesu dodávky produktu. Při zkoumání a vyhodnocování procesu je třeba brát v potaz nejen čas a efektivitu jednotlivých činností (např. analýza, vývoj, testování, nasazení), nýbrž také prodlevy a přechody mezi nimi. Ty mnohou velmi negativně ovlivňovat výslednou kvalitu celého procesu.

4.2.3 Zachování alternativních možností

Vývoj software je velmi obtížně předvídatelná činnost. Proto je vhodné být variabilní a uchovat si i alternativní možnosti. Těmi se lze ubírat v případě, že se aktuálně zvolený postup ukáže jako nesprávný či nedostačující.

Především v počátečních fázích projektu je míra nejistoty poměrně veliká. Zároveň je zde však tlak na to, aby vývoj začal co nejdříve (nebo alespoň návrh architektury a obdobné

přípravné činnosti). Tyto dva faktory mohou být příčinou nesprávného rozhodnutí (může dojít k výběru špatné možnosti). Náklady na úpravy tohoto pochybení jsou pak značné. Proto je doporučeno definovat na počátku projektu více možností a stanovit si vyhodnocovací body. V těchto bodech pak může postupně docházet k eliminaci některých možností do doby, než zůstane pouze jediná.

4.2.4 Inkrementální vývoj a integrace

Pokud klient nevidí postup v projektu průběžně, nedokáže na jeho vývoj reagovat a poskytovat cennou zpětnou vazbu. To výrazně zvyšuje riziko neúspěchu projektu, jelikož na jeho konci může dojít ke zjištění, že systém naprosto nesplňuje klientovi požadavky.

Projekt by měl být rozdělen do několika časově vymezených iterací, přičemž v každé z nich bude vytvořena jedna jeho funkční část. Poté může dojít k jejímu lepšímu otestování, vyhodnocení přínosů a klient k ní snadno může poskytnout feedback. Tímto způsobem lze projekt postupně budovat až po jeho finální dokončení a předání. Tyto integrační body také tvoří skvělou příležitost k zorganizování meetingu s vybranými stakeholdery projektu. Na setkání lze odprezentovat novou funkcionalitu, získat a zaznamenat připomínky či výhrady, shrnout aktuální stav projektu a naplánovat a odsouhlasit si další postup.

4.2.5 Navázání milníků na měřitelné body

Je žádoucí, aby zákazník průběžně odsouhlasoval postup projektu (zda dosavadní odvedená práce naplňuje jeho představy a zásadní potřeby). Proto bývají stanovovány tzv. milníky (kontrolní body). Ty jsou však často navázány na nesprávné okamžiky v rámci životního cyklu projektu. Milníky bývají definovány např. po fázi analýzy požadavků, návrhu designu, procesu implementace, testování či po dodání samotného finálního produktu. Tento přístup však není příliš vhodný.

Jakékoli vyhodnocení po fázi analýzy požadavků či návrhu designu je jen velmi obtížné a zřídkakdy přinese kýžený efekt. V takovou chvíli zákazník nedokáže správně odhadnout vhodnost řešení (nemá k dispozici žádný hmatatelný výsledek). Naproti tomu po dokončení implementace již bývá poměrně pozdě na jakékoli zásadní změny a opravy případných kritických nedostatků, které nebyly správně odhaleny v průběhu prvotních analýz. V takových situacích je obtížné nalézt pro obě strany přijatelný kompromis. Strana dodavatele systému se odvolává na schválené designy, oproti tomu zákazník deklaruje, že stávající řešení vůbec nevyhovuje jeho avizovaným potřebám.

Z výše zmíněných důvodů je nutné hledat jiný čas pro stanovení milníků. SAFe navrhuje, aby tyto body byly vždy navázány na hmatatelný výstup. Zde lze využít princip popsaný v kapitole 4.2.4 – Inkrementální vývoj a integrace. Pokud je projekt dodáván

po částech, a v každé dodávce existuje demo verze produktu, je vhodné využít těchto příležitostí na navázání milníků. Každá z verzí v sobě obsahuje část analýzy požadavků, designu, implementace i testování, proto lze ověřit všechny tyto činnosti najednou.

4.2.6 Redukce prostojů a vizualizace práce

Tento princip zajistí zrychlení průtoku úkolů systémem. Pro dosažení tohoto cíle pomáhá dodržování několika základních zásad.

První z nich je vizualizace a minimalizace činností, na kterých se pracuje. Příliš mnoho rozpracovaných úkolů působí na členy týmu poměrně negativně, jelikož vedou k neustálému přepínání pozornosti. Tento problém lze částečně vyřešit pomocí vizualizace práce (např. implementací nástroje pro správu požadavků – Jira[1], Redmine[2] apod.) nebo pomocí konceptu Kanban (princip vizualizace úkolů na separátních kartách). Do systému by také nemělo proudit více práce, než je systém chopen pojmout a zpracovat. V opačném případě se práce začíná hromadit, prioritní úkoly mohou čekat ve frontě déle, pracovníci se dostávají do stresových situací atd.

Další zásadou je redukce množství práce v jednotlivých dodávkách (zahrnující analýzu požadavků, design, implementaci, testování aj.). Čím menší a méně variabilní je práce v rámci jedné dávky, tím rychleji ji lze dokončit. Samozřejmě je nutné myslet i v ekonomických souvislostech – pokud by byla dodávka příliš malá, režijní náklady by mohly být neúměrně vysoké.

Důležitou zásadou je také řízení délky fronty úkolů. Čím je tato fronta delší, tím delší je i doba mezi zařazením úkolu do systému a jeho dokončením. Pokud je udržován přehledný a nepříliš dlouhý backlog, nově zadaný úkol projde systémem mnohem rychleji. Doba dokončení úkolu se dá také mnohem lépe předpovědět, čímž lze jednotlivé činnosti a jejich návaznosti plánovat efektivněji.

4.2.7 Rytmus a synchronizace

Práce na projektu musejí postupovat plynule a rovnoměrně. Všechny rutinní aktivity by tak měly být prováděny co nejvíce automaticky, což umožní pracovníkům vložit své kreativní úsilí do důležitých a specifických činností.

K úspěšnému dokončení projektu také napomáhá synchronizace všech zainteresovaných stran (ať už týmových nebo vnějších). Do procesu vývoje je vhodné začlenit i zákazníka či koncového uživatele. Stejně tak je nutné sjednotit společné dílčí cíle pro všechny členy týmu napříč všemi odděleními. V rámci těchto synchronizačních bodů by měl být

[1] www.atlassian.com/software/jira
[2] www.redmine.org

shrnut aktuální stav projektu, znovu se zhodnotí technické a obchodní vize a očekávání (krátkodobé i dlouhodobé) a naplánuje se další postup v rámci nadcházející fáze.

4.2.8 Apel na vnitřní motivace pracovníků

Zásadní motivační prvek pro každého člověka jsou peníze. Platí však nepřímá úměra mezi velikostí finanční odměny a její subjektivní hodnotou. Čím vyšší je finanční odměna, tím méně důležitý je pro pracovníka každý její další nárůst. Od určité hranice, která je však pro každého člověka nastavena rozdílně, začínají hrát zásadní roli zcela jiné faktory.

Pro každého člověka je velmi důležitá jistá forma seberealizace. Lidé také potřebují vědět, že jejich práce má smysl a že na druhé straně stojí subjekty, které budou její výstupy využívat a dokáží náležitě ocenit její přínosy. Pracovníci potřebují jasně vnímat, že jsou důležitou součástí týmu.

Různí lidé mají své hodnoty nastaveny rozdílně. Je tedy nutné najít mezi nimi obecnou rovnováhu, případně je přizpůsobovat potřebám každého pracovníka individuálně. Správně motivovaný a spokojený pracovník pak odvede kvalitní práci.

4.2.9 Decentralizace rozhodování

Pokud je neustále nutné eskalovat každodenní rozhodnutí, dochází k mnoha zbytečným prostojům a zdržením. Zároveň se při tomto procesu mohou ztrácet některé důležité informace, které jsou potřebné k učinění optimálního rozhodnutí. Proto by se mělo co nejvíce rozhodnutí decentralizovat (vyjma těch strategických, ekonomicky zásadních a dlouhodobých či nezvratných).

Každodenní, často se opakující, spěchající či nízkoúrovňová rozhodnutí by se měla provádět decentralizovaně – bez nutnosti eskalace. Měla by být učiněna pracovníkem, který má všechny potřebné informace nezprostředkovaně a zná přesný kontext problému. V případě, že zodpovědný pracovník nedokáže problém vyřešit samostatně, je vhodné jej konzultovat či předat jinému pracovníkovi na stejné úrovni. V případě nutnosti jej lze posunout o jednu úroveň výše.

4.3 SAFe tým

Metodika SAFe definuje pojem ART (*Agile Release Train*). Jedná se o soubor několika agilních týmů, které fungují jako celek („virtuální organizace“) – průběžně dodávají přírůstky produktu a soustředí se na společné cíle. Počet lidí v ARTu by se měl pohybovat

mezi 50 a 125 členy. Ti pracují na definici nových funkcionalit, jejich implementaci, testování a nasazení, čímž průběžně generují hodnotu. Je důležité, aby v rámci ARTu byly zastoupeny všechny role a každý ART tak mohl pracovat samostatně a byl plně samořídící.

I každý tým v ARTu by měl být schopen fungovat zcela samostatně a měl by čítat 5–11 členů. V otázkách týmu se SAFe inspiruje základními vlastnostmi metodik Scrum, XP (Extreme Programming) a Kanban. Tým by měl být samořídící, samoorganizovaný a samostatně zodpovědný za všechny základní činnosti – definování a návrh funkcionalit, jejich implementaci, otestování i nasazení. Framework SAFe definuje následující týmové role:

- Člen vývojového týmu – člověk, který patří do skupiny analytiků, vývojářů, testerů či ostatních nezbytných specialistů. Ti společně spolupracují na vytvoření funkčního přírůstku systému.
- Product Owner – osoba zodpovědná za týmový backlog. Provádí definici úkolů, jejich prioritizaci i finální akceptaci. Tvoří pomyslnou vazbu mezi zákazníkem a agilním týmem a jedná se tak o klíčovou roli ve vztahu k zajištění kvality produktu.
- Scrum Master – člen týmu, který zastává především roli týmového lídra a agilního kouče. Organizuje týmové události, odstraňuje překážky a odstiňuje tým od nežádoucích vnějších vlivů. Tím poskytuje podporu celému zbytku týmu, který tak může veškerou svou pozornost věnovat aktivitám, které generují přidanou hodnotu.

Takto sestavené agilní týmy by měly dodržovat následující postupy a aktivity:

- Plánování iterace – určení iteračních cílů a prvotní tvorba backlogu.
- Provedení iterace – soustavná činnost, kdy celý agilní tým pracuje na vytvoření inkrementálního přírůstku systému v rámci jasně časově definované iterace. To zahrnuje i každodenní standupy, které slouží k lepší koordinaci týmu, odhalení problémů a zhodnocení průběhu iterace.
- Zhodnocení iterace – dochází k posouzení iterace na základě zhodnocení přírůstku systému.
- Retrospektiva – na základě zhodnocení iterace se identifikují činnosti a postupy, které mohou být v rámci příštích iterací zlepšeny.
- Ladění backlogu – v průběhu iterace musí docházet k průběžným kontrolám plánu a k případným úpravám úkolů v backlogu.

Ve výše popsaných činnostech a postupech pomáhají členům týmu následující týmové artefakty:

- Story – tento konstrukt umožňuje popis uživatelských požadavků. Jedná se o klíčový artefakt agilních metodik, který obsahuje stručný popis funkcionality v lidském jazyce. Takto zpracované požadavky pak slouží jako základ pro implementaci funkcionality v rámci iterace. Jejich tvorba, prioritizace a akceptace je plně v kompetenci

Product Ownera, přičemž ostatní členové agilního týmu by měli mít také možnost přidávat své poznámky, postřehy či výhrady.

- Story aktivátor (*Enabler story*) – zahrnuje přípravy pro umožnění implementace jisté funkcionality.
- Cíle iterace – obecné představy o tom, co má být výsledkem iterace. Jsou výstupem fáze plánovaní a umožňují lepší orientaci a navigaci v průběhu iterace (je pak zřejmé, kam mají práce směřovat).
- Týmový backlog – obsahuje jednotlivé úkoly v rámci dané iterace. Skládá se z uživatelských stories a jejich aktivátorů.

4.4 SAFe program

Stejně jako je potřeba řídit jednotlivé týmy, je také nezbytné přistupovat obdobným způsobem k celým ARTům. Je nutné, aby všechny týmy, stakeholdeři i ostatní subjekty pracovali společně se stejnou vizí a cílem. Všechny týmy, role a aktivity by tak měly projektu generovat hodnotu průběžně a inkrementálně ji navyšovat. Práce na každém novém přírůstku systému by měly být časově ohraničené a typicky bývají 8–12 týdnů dlouhé.

Řízení na programové úrovni je zajišťováno pomocí následujících rolí, které sjednocují lidi za účelem dosažení společných cílů a pomáhají koordinovat ARTy:

- Systémový architekt – jednotlivec či malý tým lidí, kteří plně chápou a ovládají principy systémového myšlení. Tato role je zodpovědná za návrh celkové architektury systému a jednotlivých rozhraní, definici mimofunkčních požadavků či specifikaci klíčových funkcionalit.
- Produktový manažer – role zodpovědná za programový backlog. Prezentuje názory a požadavky klienta týmům. Jedná se tedy o primární bod komunikace se zákazníkem a Product Owners, což umožňuje lepší ujasnění potřeb klienta, definici systémových požadavků a jejich finální ověřování.
- Inženýr dodávek (*Release Train Engineer*) – stará se a optimalizuje hladký a konstantní průběh dodávek. Má hlavní slovo při plánování prací na jednotlivých produktových přírůstcích a účastní se ověřování kvality procesu dodávek.
- Vlastník byznysu – skupina stakeholderů, kteří průběžně dohlížejí na to, aby vyvíjený systém splňoval základní požadavky klienta (tzv. *fitness for purpose*).

Koordinaci ARTů usnadňuje několik základních programových aktivit:

- Plánování programového přírůstku – stěžejní činnost probíhající v rámci ARTu, která umožňuje nastavit a ujasnit si postup v další iteraci. Jedná se především o sestavování a prioritizaci programového backlogu a o stanovování cílů při implementaci přírůstku. Součástí může být také definice kritických činností a rizik.

- Demo systému – funkční a plně integrovaná část systému, která obsahuje funkcionalitu dodanou v rámci poslední iterace. Demo poskytuje všem stakeholderům prostředek k objektivní verifikaci dodaného projektového přírůstku. Na jeho základě lze snadno získat rychlý feedback, který je velice hodnotný.
- Kontrola a přizpůsobování (*Inspect & Adapt*) – činnost, jejímž hlavním účelem je ověření a vyhodnocení stavu projektu. Stejně jako demo systému by i tato činnost měla být prováděna po každém dokončení produktového přírůstku. Na jejím základě lze poté navrhovat úpravy a vylepšení projektových procesů. Měla by se skládat z interního předvedení dema systému, jeho kvantitativního ohodnocení, retrospektivy, a nakonec z workshopu, z něhož vyplynou řešení odhalených problémů.

Z programových aktivit a činností pak plynou následující artefakty:

- Feature – jedná se dílčí funkcionalitu, která naplní konkrétní požadavek zákazníka. Měla by obsahovat název, popis přínosů a akceptační kritéria.
- Capability – některé features jsou tak rozsáhlé, že je nutné je rozdělit mezi více ARTů. K jejich popisu slouží capability, která má obdobný formát, avšak umožňuje takto široké features sjednotit a zastřešit.
- Epic – velmi obecný popis potřeby některého ze stakeholderů. Na jejím základě jsou pak definovány jednotlivé features.
- Programový backlog – slouží k uchovávání všech features, capabilities a epics v rámci každého ARTu. Také obsahuje tzv. *enabler features*, které jsou základem pro přípravu architektury. Na jeho tvorbě se podílejí všechny programové role i klient.
- Programový Kanban – vizualizační metoda, která umožňuje lepší koordinaci a orientaci v programovém backlogu. Tím lze lépe organizovat průběh prací v rámci iterace a odhalit tak případná úzká hrdla v procesu.
- Cíle systémového přírůstku – popis obchodních a technických cílů, kterých musí ART dosáhnout během následující iterace. Jedná se o výsledek aktivity plánování programového přírůstku.
- Příprava architektury – veškeré komponenty, kód a infrastruktura potřebná k hladkému průběhu tvorby následujícího produktového přírůstku. Je tvořena na základě enablerů (aktivátorů).

4.5 SAFe portfolio

Tato úroveň frameworku SAFe obsahuje osoby a procesy, které jsou nezbytné k vytvoření systému a řešení, jenž umožní společnosti splnění jejích strategických záměrů. Obsahuje tedy principy, praktiky a role pro střední a vyšší management společnosti. To jim umožňuje efektivně řídit hodnotové toky v rámci celé organizace. Zahrnuty jsou také rady pro for-

mování strategie a financování investic, které poskytnou společnosti optimální návratnost (ROI – *Return of Investment*). Velké společnosti mohou obsahovat několik různých instancí této úrovně.

Na úrovni portfolia jsou následující role, které mají velkou míru odpovědnosti a pomáhají řídit a koordinovat hodnotové řetězce:

- Management štíhlého portfolia – tato funkce reprezentuje osoby, které jsou zodpovědné za rozhodování a finance v rámci SAFe portfolia.
- Epic Owner – osoba zodpovědná za koordinaci podnikových epics.
- Podnikový architekt – osoba zodpovědná za epic aktivátory. Pomáhá naplňovat strategické cíle organizace z architektonického a technologického hlediska.

K popisu strategických záměrů společnosti slouží následující artefakty:

- Podnikový epic – zachycuje nové požadavky podniku, které lze realizovat pomocí koordinace mezi hodnotovými řetězci.
- Epic aktivátor (*Enabler epic*) – reprezentuje architektonické nebo jiné technologické potřeby, které jsou nezbytné k dokončení nějaké funkcionality.
- Strategické téma – specifický a detailně rozepsaný cíl, kterého je třeba dosáhnout k naplnění organizační strategie.
- Backlog portfolia – nejvyšší úroveň backlogů. Obsahuje schválené podnikové epics a jejich aktivátory.

4.6 Případové studie

Na oficiálních stránkách frameworku SAFe je k dispozici více než 50 případových studií. K nim jsou také připojeny krátké příběhy o dalších projektech, které dokázaly framework úspěšně implementovat. Jednotlivé implementace SAFe jsou z mnoha různých odvětví. Jedná se např. o projekty pro vládní sektor, zdravotnictví, finanční odvětví, výrobu či technologické a telekomunikační společnosti. V následující části jsou shrnuty poznatky z případových studií pro bankovní společnost Capital One a pro technologickou společnost Intel. [16]

4.6.1 Capital One

Projekt potřeboval zlepšit efektivitu reakce na požadavky trhu. To znamenalo transformovat způsob dodávky software směrem k agilnějšímu přístupu, navíc pro rozsáhlé projekty. Framework SAFe byl zvolen, jelikož poskytoval jasné a jednoznačné pokyny, kvalitní podporu, tréninky a rady expertů. Vyvíjené produkty byly poměrně rozsáhlé, takže byla potřeba vzájemná týmová interakce a společné plánování. V této koordinaci implementace SAFe výrazně pomohla.

Došlo ke zvýšení angažovanosti zaměstnanců o 15–20 % a úspěšně byl integrován agilní přístup v rámci celé organizace. Také byla přehodnocena strategie využívání externích aplikací, místo čehož došlo k posunu směrem k jejich internímu vývoji. [17]

4.6.2 Intel

Projektové řízení shledávalo agilní přístup velmi atraktivním, avšak škálování standardního Scrumu se s růstem projektu stávalo stále obtížnějším úkolem. Intel potřeboval dosáhnout kontinuální inovace produktu, přičemž musely být efektivně řízeny náklady a zachována kvalita. SAFe pro ně představoval prokazatelně fungující a veřejně známý framework, který má navíc jasně definované a popsané role a artefakty vycházející ze štíhlého a agilního přístupu. Také bylo důležité, že SAFe popisuje všechny úrovně organizace.

Výsledkem implementace frameworku bylo 65% navýšení množství dodaných produktů při zachování stejných pracovních kapacit. Vše se stalo transparentnější a přehlednější pro všechny zúčastněné osoby. Množství změn v rozsahu projektu bylo zredukováno na méně než 5 %. [18]

5 DISCIPLINED AGILE DELIVERY

Prvotní koncept a základní myšlenky frameworku Disciplined Agile Delivery (DAD) vznikaly pod záštitou společnosti IBM. Tento framework vznikal proto, že velké množství společností mělo problémy s adopcí agilních principů pro rozsáhlejší projekty. V současné době práce na frameworku stále probíhají. Soustředí se však na projekt s vyšším nadhledem a komplexněji, nikoliv výhradně na jeho dodávku. Framework tak bývá stále častěji označován pouze pojmem Disciplined Agile (DA).

Tato kapitola obsahuje ucelený popis frameworku DAD. Jako hlavní zdroj informací byla využita především oficiální knižní publikace o DAD [19]. Další informace byly získány zejména z oficiálních webových stránek frameworku [20] a z veřejně dostupných případových studií [21].

Jedním ze základních důvodů, které stojí za vznikem DAD, je přespříliš doslovné dodržování moderních agilních principů. V minulosti byly metodiky příliš byrokratické a přehnaně zaměřené na dokumentační část. S nástupem moderních trendů a agilních metodik se však vše otočilo do opačného extrému – v některých projektech neexistovaly žádné artefakty kromě samotného kódu. Je tedy potřeba přistupovat k projektům více disciplinovaně. DAD se tedy snaží najít optimální rovnováhu mezi těmito přístupy (samozřejmě jsou však preferovány základní aspekty agilního manifestu – viz kap. 2.2).

Další motivací pro vznik DAD bylo to, že agilní metodiky (např. Scrum nebo XP) fungují plnohodnotně spíše pro menší typy projektů. Z tohoto důvodu vznikl komplexní hybridní framework DAD, který kombinuje poznatky a postupy z mnoha různých zdrojů a přizpůsobuje je potřebám rozsáhlých projektů. Základem je samozřejmě Scrum, ale použity jsou také některé praktiky známé z Agilního modelování, XP, RUP (*Rational Unified Process*) atd.

DAD se zabývá celým životním cyklem vývoje produktu – od prvotní myšlenky, přes celý proces jeho dodávky, až po nasazení a předání klientovi (včetně následné podpory výsledného systému). Snahou frameworku je také vyplnit všechny nedostatky a prázdná místa, která nejsou detailně popsána v jiných agilních metodikách (např. modelování, dokumentace nebo strategie řízení projektu).

Zcela stěžejní charakteristikou DAD je značná míra nabízených alternativ u většiny postupů a popis jejich jednotlivých výhod a nevýhod. To umožňuje uzpůsobení a konfiguraci postupů pro konkrétní potřeby organizace či projektu.

Následuje podrobný popis klíčových charakteristik frameworku, popis týmových rolí a rozbor jednotlivých DAD fází (zahájení, konstrukce a předání). Závěrem jsou shrnuty 2 případové studie.

5.1 Klíčové charakteristiky

Framework DAD má několik základních a důležitých charakteristik. Ty zpravidla vyplynuly z již existujících metodik a přístupů. V následujících podkapitolách jsou jednotlivé charakteristiky stručně popsány a shrnuty.

5.1.1 Lidé na prvním místě

DAD razí myšlenku, že lidé a způsob jejich spolupráce jsou hlavním faktorem, jenž přímo ovlivňuje kvalitu a úspěch dodávaného IT řešení. Jedná se také o první zásadu agilního manifestu (viz kap. 2.2). Pokud je tým složen z kvalitních členů, je nejlepší nechat jim volnost, aby většinu problémů vyřešili samostatně. Členové DAD týmu by měli mít následující tři základní vlastnosti:

- Disciplína – měli by se podílet na činnostech, které dokáží zvládnout s co největší mírou efektivity (rychle, ale zároveň také kvalitně).
- Samoorganizace – dokáží odhadnout a samostatně si naplánovat svou nadcházející práci. Následně musí umět spolupracovat s ostatními členy týmu na jejím úspěšném dokončení.
- Uvědomění – musejí umět identifikovat takové postupy, které pro ně fungují. Také je důležité dokázat odhalit ty činnosti, které jim příliš nevyhovují a negativně ovlivňují jejich práci. Následně se ze zjištěných skutečností musejí poučit a tyto postupy náležitě upravit.

DAD také podporuje to, aby se lidé v týmu soustředili na více dílčích úkolů a zastávali různé role. Samozřejmě, každý z členů týmu by se měl specializovat na určité činnosti. Je však vhodné, aby jejich znalosti přesahovaly i do jiných oblastí. Tím dochází k přirozenému předávání vědomostí a usnadňuje to komunikaci v rámci celého týmu.

5.1.2 Orientace na proces učení

Učení probíhá na několika různých úrovních. První z nich se vztahuje k identifikaci požadavků zákazníka (podle DAD se jedná o tzv. učení na úrovni domény). Toho lze dosáhnout např. pomocí inkrementálních dodávek, předvádění dema systému nebo aktivního začlenění stakeholderů do průběhu projektu. Druhou úrovní je učení na úrovni procesu. To zahrnuje postupné zlepšování členů týmu, týmu jako celku i procesů na úrovni celé společnosti. K tomuto účelu pomáhají např. retrospektivy. Poslední úrovní je učení na technologické bázi. To bývá v IT velmi přirozené, jelikož pracovníci v této oblasti jsou většinou ochotni používat nové technologie a sami aktivně sledují moderní trendy, nástroje a techniky.

Rozdílem mezi DAD a jinými standardními agilními metodikami (např. Scrum či XP) je to, že se soustředí nejen na procesní aspekty v učení (na úrovni projektu), ale také na organizaci jako celek. Je tedy doporučováno, aby zjištěná ponaučení byla předávána i na vyšší úrovně. Management společnosti se tak také může učit a zlepšovat celopodnikové procesy.

5.1.3 Agilní přístup

Framework DAD využívá principů agilního přístupu a jeho základních zásad. Také vyznává základní hodnoty zmíněné v agilním manifestu (více v kap. 2.2).

5.1.4 Životní cyklus dodávky je orientován na cíl

DAD je zaměřen na cíl. To znamená, že je zde snaha eliminovat veškeré nadbytečné činnosti, které výslednému produktu negenerují žádnou reálnou přidanou hodnotu. Nemá tedy smysl např. vytvářet dokumenty, které nejsou nezbytné. Tento přístup je aplikován ve všech částech životního cyklu celé dodávky.

Scrum se zabývá výhradně konstrukční fází projektu. DAD to pojímá komplexněji a definuje 3 základní fáze – zahájení, konstrukce a předání (jsou tedy definované obdobně jako v metodice RUP). Každá z těchto fází je opět dělena do 3 dalších částí, které jsou v podstatě totožné, pouze jsou jinak nazývány – koordinace, provedení a uzavření. Podobně je pak přistupováno i k činnostem prováděným v rámci každého pracovního dne.

5.1.5 Hybridní framework

Pojem hybridní framework indikuje, že se jedná o koncept vytvořený na základě různých strategií a přístupů. DAD tedy přejímá některé principy a postupy z rozličných zdrojů [19]:

- Scrum – stěžejní koncept frameworku DAD. Přejat je např. prioritizovaný zásobník s činnostmi (backlog), role Product Ownera reprezentujícího zájmy stakeholderů nebo tvorba funkčního řešení v rámci každé iterace.
- XP – odsud je přebrána zejména průběžná integrace, refactoring, programování řízené testy (unit testy) či kolektivní vlastnictví kódu (každý vývojář smí upravovat libovolnou část kódu).
- Agilní modelování – jedná se o metodiku určenou k efektivnímu modelování a tvorbě dokumentací. To zahrnuje vizualizaci požadavků a architektury, návrh iterace, kontinuální tvorbu dokumentace aj.
- RUP – DAD přebírá také některé postupy, které jsou v souladu s agilním myšlením, z metodiky RUP. Přejato je např. stanovování milníků či rozdělení projektu do fází.

Také je kladen důraz na brzký návrh architektury (tzv. PoC – *Proof of Concept*) a snaha o co největší redukci rizika v počátečních fázích projektu.

- Agilní data – jedná se o soubor postupů, jak pracovat s daty v databázi. Zahrnut je např. databázový refactoring, databázové testování či datové modelování.
- Kanban – tato metoda se používá především za účelem limitace současně probíhajících činností a k přehlednější vizualizaci práce.

5.1.6 IT řešení místo pouhého software

DAD popisuje, že IT společnosti by ke své činnosti neměly přistupovat pouze jako k vývoji software. Místo toho by jejich hlavním záměrem mělo být poskytnutí komplexního IT řešení, které přináší reálnou přidanou hodnotu jejich stakeholderům. Mělo by také naplňovat jisté ekonomické, kulturní a technické aspekty.

5.1.7 Zaměření na organizaci

Každý pracovník v DAD týmu by měl dělat nejlepší možná rozhodnutí pro organizaci jako celek. To zahrnuje např. používání již dokončených a otestovaných funkcionalit, postupů, nástrojů či zdrojů dat. V případě potřeby je vhodné všechny tyto postupy a nástroje systematicky vylepšovat tak, aby byla průběžně vytvářena a aktualizována stávající báze.

Všechny nabyté znalosti a zkušenosti by se také měly v rámci organizace náležitě šířit. Proto DAD doporučuje zavedení interních diskuzních fór, prezentací, konferencí a školení od seniorních členů společnosti. Důležitý je také průběžný tok informací, které potom slouží různým vrstvám v rámci organizace k vyhodnocování aktuálního stavu projektu a k učinění kvalifikovaných a informovaných rozhodnutí.

5.1.8 Orientace na hodnotu a eliminaci rizik

DAD se v maximální možné míře snaží minimalizovat rizika a kontinuálně navyšovat hodnotu produktu. To zahrnuje průběžné kontroly, při kterých je zjišťováno, zda byla daná funkcionalita vyvinuta podle představ a potřeb klienta. V rámci tohoto přístupu neprobíhá prioritizace backlogu pouze na základě největší přidané hodnoty, ale také na základě rizik. Snahou je tedy zpracovat ty nejrizikovější činnosti co nejdříve, aby případné problémy a komplikace nebyly odhaleny až ve finálních fázích projektu.

Agilní metodiky obecně doporučují nejen tvorbu použitelného řešení v každé iteraci, ale také jeho předvedení stakeholderům, čímž je dosaženo jejich aktivního zapojení do průběhu projektu. Stakeholdeři by se také měli mít možnost účastnit např. fází modelování a dalších inicializačních činností.

DAD tyto činnosti navíc rozšiřuje a definuje několik potenciálních milníků:

- Shoda mezi stakeholdery – měla by panovat po dokončení zahajovací fáze. Slouží k verifikaci pochopení potřeb klienta.
- Důkaz funkčnosti architektury – tento milník by měl být umístěn na počátek konstrukční fáze. Je nutné vědět, že architektura je navržena správně a bude v budoucnosti funkční.
- Dostatečná životaschopnost řešení – průběžná kontrola toho, zda dosavadní práce na projektu odpovídají potřebám klienta. Měla by probíhat každé 2–3 měsíce.
- Připravenost produktu – bývá na konci konstrukční fáze a předchází fázi nasazení do produkce. V jejím rámci je vyhodnocováno splnění akceptačních kritérií.
- Připravenost pro produkci – odehrává se před koncem fáze nasazení. Stakeholdeři akceptují produkt pro produkci a jsou spokojeni s příslušnými procedurami a dokumenty.
- Spokojenost stakeholderů – produkt je nasazen, kompletně začleněn do provozu a klient je s výsledkem spokojen.

5.1.9 Škálovatelnost

DAD si zakládá na tom, aby se nejednalo o soubor striktních postupů, které jsou totožné pro všechny projekty – bez ohledu na jejich velikost, komplexnost nebo technickou složitost. Také je důležité brát zřetel na geografické uspořádání týmu, organizační strukturu, postupy ve společnosti apod.

5.2 DAD tým a týmové role

DAD se skutečně velmi soustředí na první princip agilního manifestu – lidé a jejich interakce jsou na prvním místě a procesy a nástroje jsou až za nimi. Tento framework podporuje přiřazení více rolí každému člověku. Pracovníci by tak měli mít přehled i o jiných oblastech v projektu, než je jejich primární specializace. Zároveň je možné, aby se role člověka v průběhu projektu měnila, což umožňuje vyšší míru flexibility. Velmi důležitou myšlenkou je to, že všichni lidé a všechny role jsou na stejné úrovni. Neexistuje zde žádná hierarchie.

DAD definuje dvě základní skupiny rolí. První skupinou jsou tzv. primární role. Tyto role pokrývají veškeré činnosti, práva a zodpovědnosti, které musí být obsaženy v každém projektu. Jedná se o skutečně stěžejní role, které však ve všech případech nemusí být úplně dostačující. Druhou skupinou jsou sekundární role, které jsou více specializované a mohou být potřebné pouze v určitých konkrétních částech projektu.

Většina rolí je založena na metodice Scrum. Mezi primární role patří:

- Člen týmu – jedná se o základní jednotku, která provádí analýzu, testy, programování, plánování apod. Není potřeba, aby každý člen obsáhl všechny tyto činnosti. Je zcela dostačující, když zná alespoň nějakou jejich podmnožinu. Členové se podílejí na identifikaci úkolů, jejich plnění, odhadech a zaznamenávání jejich stavu. Tým by měl být samoorganizovaný a měl by systematicky spolupracovat.
- Týmový lídr – ekvivalent Scrum Mastera ve Scrumu. Odstraňuje překážky, které zabraňují týmu ve vykonávání jeho základních činností, dohlíží na plnění projektových cílů a vizí, odstiňuje tým od nežádoucích externích vlivů apod.
- Vlastník produktu (Product Owner) – tato role primárně slouží pro reprezentaci potřeb a požadavků zákazníka. Jeho plnou kompetencí je prioritizace produktového backlogu. Na rozdíl od Scrumu však prioritizuje i další projektové činnosti (nasazování, školení, prezentace apod.). Každý tým by měl mít vlastního Product Ownera. Jeho sekundární zodpovědností je prezentace výsledků práce stakeholderům. Také je to první osoba, na kterou se členové týmu obracejí v případě dotazů či nejasností.
- Vlastník architektury – tato role je přidána oproti Scrumu, jelikož návrh architektury je klíčová činnost každého projektu. Podle DAD však není nutné formálně přidělit tuto roli konkrétnímu členovi (především u menších projektů). Může jí souběžně zastávat více členů nebo třeba týmový lídr. Člověk s touto rolí vykonává obdobné činnosti jako kterýkoli jiný člen týmu, navíc však vede technické a technologické debaty ohledně architektury.
- Stakeholder – podle DAD je i tato skupina přímou a nedílnou součástí týmu. Stakeholder je každý, kdo je ovlivněn výsledkem projektu. DAD je dělí do 4 kategorií: koncový uživatel, management, partneři a servisní podpora.

Mezi sekundární role pak patří:

- Experti a specialisté – tito lidé slouží jako určitá podpora (či doplněk) Product Ownera. Ten totiž u komplexnějších projektů zpravidla není schopen obsáhnout všechny potřebné oblasti a nemůže znát všechny detaily. Proto je možné začlenit do týmu po určitou dobu i experty na určitou oblast. Může se jednat o oblasti technologické (databázový administrátor, grafik, GUI či UI/UX specialista, expert na nějaký konkrétní framework apod.) nebo obchodní (daně, logistika, výroba atd.) – takový specialista je zpravidla vybírán z řad stakeholderů. Do této kategorie také dále patří např. programoví manažeři, DAD specialisté či agilní kouči.
- Nezávislý tester – ačkoli většina testovacích činností probíhá v rámci DAD týmu, občas je žádoucí, aby existovalo i paralelní testování na jiné úrovni. Jedná se o tým lidí, kteří testují přímo za stranu stakeholdera. Stojí tedy mimo DAD tým a fungují na něm zcela nezávisle.

- Integrátor – u velkých týmů a rozsáhlých projektů bývá často problém s integrací dílčích částí systému. U malých projektů může tuto činnost zastávat např. vlastník architektury. Čím je však projekt větší, tím vyšší bývají i nároky na jeho integraci. V takovém případě je pak vhodné delegovat tuto roli alespoň jednomu člověku. Ten pak velmi často funguje s nezávislým týmem testerů a spolupodílejí se na provádění integračních testů.

DAD počítá s tím, že každý projekt má odlišné nároky na budování a sestavování týmu. Proto v této oblasti nechává velkou volnost a nabízí různá řešení pro různě velké týmy. Popisuje tak ideální a doporučené složení pro malé (do 15 lidí), střední (10–40 osob) a velké týmy (30 a více členů).

5.3 DAD fáze

Framework DAD rozděluje projekt do 3 základních fází – zahájení, konstrukce a předání (viz příloha B). V rámci každé z nich je prováděno několik činností, které vedou k jejímu naplnění. V této kapitole jsou jednotlivé fáze a jejich specifické činnosti podrobně popsány.

5.3.1 Fáze zahájení

Tato fáze bývá nazývána také jako inicializační či nultá iterace. Jedná se o fázi předcházející samotnému započetí konstrukce a programování projektu. Jejím cílem je ujasnění si potřeb zákazníka, nadefinování projektové vize, identifikace dostupných technických řešení a naplánování budoucího postupu. Dále je pak nezbytné zajistit financování a připravit veškeré pracovní prostředí. Současně probíhá sestavování týmu. Nakonec je nutné, aby klíčoví stakeholdeři zhodnotili nadefinované postupy a schválili zvolenou strategii.

Po čas této fáze je doporučeno dodržování několika základních praktik, které jsou již ověřeny praxí. Jedná se např. o její délku. Ta by neměla přesahovat jeden měsíc, přičemž ideální stav jsou 2 týdny. Opačný případ indikuje přílišnou byrokracii či špatně nastavené vnitřní procesy. Další doporučenou praktikou je definice pohyblivých odhadů – náklady a časový plán je vhodné pojmout formou intervalů (např. v rozpětí 5 % do obou směrů). Tím je dosaženo lepší flexibility a projekt tak lze lépe řídit a přizpůsobovat jej v rámci tzv. železného trojúhelníku (rozsah, čas a náklady). Poslední pozitivní praktika se týká dokumentace. Ta by měla být minimální, avšak dostačující. Měla by obsahovat především základní modely, diagramy a další návrhy.

DAD také varuje před rozšířenými negativními praktikami v zahajovací fázi. Mezi ně patří např. nedostatečně připravené prostředí, chybějící článek zkušeného agilního řízení, přílišné a nadbytečné zásahy managementu shora, předčasný přechod do fáze konstrukce,

přehnaně detailní plánování nebo paralýza analýzou (nesmyslná snaha o perfektní a detailní dokumentaci).

Inicializační projektová fáze se podle DAD skládá z několika základních činností. V následujících kapitolách je zmíněn jejich stručný popis.

Identifikace vize projektu

Projektová vize by primárně měla odpovídat na otázku, jaké jsou cíle projektového týmu. Dále definuje rozsah projektu a jeho fáze. Její součástí může být také nástin technické strategie a orientační plán práce. Může být přiložena i studie proveditelnosti.

Vize primárně slouží jako podklad při několika základních situacích. První z nich je její využití při zhodnocování finální funkcionality produktu. Může být tedy použita jako základ pro definici výstupních kritérií. Dále pak usnadňuje týmu orientaci v projektu a pomáhá jej udržovat na správné cestě k dokončení požadovaného řešení. Také často slouží jako podklad pro získání finančních prostředků pro tým. Klíčoví stakeholdeři a projektoví sponzoři by se měli s nadefinovanou vizí plně ztotožňovat.

DAD také popisuje různé přístupy, jak zachytit a sepsat projektovou vizi. Dále specifikuje několik možností, jak zapojit stakeholdery do její tvorby a jak dosáhnout akceptace vize z jejich strany.

Prvotní definice rozsahu

V rámci této činnosti není potřeba vytvářet podrobný, kompletní či finální popis projektu. Ve skutečnosti by to bylo spíše kontraproduktivní, jelikož časové nároky by byly příliš vysoké. Cílem je spíše vytvoření určité obecné projektové osnovy, kterou bude možné v budoucnu následovat.

Díky této činnosti lze lépe odhadovat obsahovou, časovou i finanční náročnost. Také díky ní lze odhalit spoustu problémů hned na začátku projektu. Pomocí různých typů modelů lze lépe rozpoznat některé souvislosti a pohlédnout na jisté části projektu z různých perspektiv.

DAD nabízí několik přístupů souvisejících s výběrem vhodné míry detailu. Také definuje několik typů použitelných modelů (např. model případů užití, doménový model, procesní model nebo nejrůznější typy UI modelů). Stejně tak se zabývá i různými způsoby pořizování těchto modelů (pomocí interview, formálních schůzek, méně formálních workshopů apod.). Dále je potřeba rozhodnout, jak bude sestavován a prioritizován zásobník s úkoly a činnostmi. Nakonec se framework zabývá i strategiemi pro přístup k mimofunkčním požadavkům.

Určení úvodní technické strategie

V úvodu projektu je potřeba mít obecnou představu o tom, jak bude systém konstruován. Jedná se např. o volbu technologií, ve kterých bude systém vyvíjen a o dostupné frameworky, knihovny či další nástroje, které by mohly být pro potřeby projektu využity. Úvodní modelování pozitivně přispívá ke zlepšení produktivity, redukuje technická rizika a snižuje čas vývoje.

DAD opět poskytuje různé pohledy, jak v této fázi postupovat. Jedná se o výběr optimální úrovně detailu modelů, výběr správných typů modelů a volbu přijatelných strategií pro jejich vytváření. Návrh architektury by měl být diskutován také s podnikovým architektem. V počátcích konstrukční fáze by měl být tento návrh podpořen fungujícím kódem.

Plánování nasazování a dodávek

DAD doporučuje zamyšlení se nad plánováním nasazování již v inicializační fázi. Cílem však nejsou návrhy na konkrétní technické řešení, ale spíše rozvržení zdrojů (časový plán, náklady, alokace lidí). Jedná se např. o určení frekvence nasazování. To může probíhat např. čtvrtletně, měsíčně, týdně nebo klidně i denně. V této souvislosti se samozřejmě rozlišuje mezi interními a externími nasazeními.

Příprava týmu a pracovního prostředí

Vedení týmu musí spolupracovat se sponzory projektu tak, aby byla plně zajištěna připravenost prostředí a došlo tím k maximalizaci efektivity týmu. V některých organizacích může docházet k alokaci finančních i lidských zdrojů až po dokončení inicializační fáze (po odsouhlasení vize klíčovými stakeholdery). V takovém případě je nezbytné provést tyto činnosti ihned po započetí konstrukční fáze.

V rámci této činnosti popisuje DAD různé postupy při formování projektového týmu, při výběru sady nástrojů, organizaci fyzického pracovního prostředí (prostory, tabule, projektor apod.) a organizaci virtuálního pracovního prostředí i pro distribuované týmy (jednotné online úložiště pro ukládání artefaktů, komunikační nástroje atd.). Dále je pak doporučeno zamyslet se nad tzv. vizuálním managementem – grafy, rozvrhy, dashboardy, plán dovolených atp. To umožňuje lepší přehled o stavu projektu i pro lidi mimo něj.

5.3.2 Konstrukční fáze

DAD se v rámci postupů v této fázi citelně inspiroval praktikami Scrum a XP. Po čas této fáze je celý projekt konstruován v sérii po sobě jdoucích dílčích fází (iterací, sprintů). Tyto iterace musejí být jasně časově vymezeny. Primárním cílem konstrukční fáze je vytvoření

použitelného řešení, které přinese stakeholderům přidanou hodnotu. Dalšími cíli této fáze jsou např. brzké vytvoření důkazu o funkčnosti a vhodnosti použité architektury, přizpůsobování se a reakce na změny klientských požadavků nebo průběžný posun vstříc nasaditelné verzi. Po celou dobu trvání konstrukční fáze by měl být kladen důraz na zachování či vylepšení nastavené úrovně kvality kódu.

Stejně jako u předchozí fáze jsou také zde definovány doporučené i negativně hodnocené praktiky. Do první skupiny patří např. možnost demonstrace produktového přírůstku po dokončení každé iterace nebo aktivní participace členů týmů na společných úkolech v případě, že dokončí svá zadání s předstihem. Dále je pak vhodné, aby se koncové datum iterace nikdy neposouvalo. Poslední doporučovanou praktikou je poskytnutí možnosti kterémukoli stakeholderovi v domluvený čas přijít a vyžádat si předvedení aktuálního stavu projektu.

Mezi praktiky, kterým by se měl projektový management snažit vyhnout, patří např. přílišná rozsáhlost a nekonsolidovanost iteračního backlogu, nedostatečné předcházení riziku (ignorace riskantních činností) nebo neověření funkčnosti architektury. Dále by se měli všichni členové týmu soustředit pouze na daný projekt – nemělo by docházet k jejich sdílení s ostatními projekty. Také by nemělo docházet k častému a systematickému opomíjení činností v rámci iteračního plánování a k jejich následnému zařazování do backlogu v průběhu iterace. To negativně přispívá k vytváření chaosu a k narušení krátkodobých plánů. Indikací velmi zásadních problémů v projektu je také to, pokud se s každou dokončenou iterací zvyšuje počet nalezených defektů. To svědčí o nedostatečné kadenci oprav a vypovídá to o snaze přidávat nové funkcionality na úkor udržení kvality.

DAD rozděluje každou konstrukční iteraci na 3 základní části – inicializační, vývojová a uzavírací. Následuje stručný popis všech těchto částí.

Inicializace konstrukční iterace

V rámci této části dochází k podrobnějšímu rozkladu a definici činností, které byly naplánovány v rámci fáze zahájení pro danou iteraci. Na základě těchto činností dochází k plánování každodenních prací. Je důležité, aby se této části účastnil celý tým a plánování nadcházející iterace tak probíhalo na základě týmové spolupráce a společného úsilí.

Jednou z hlavních činností je zde tedy plánování iterace. V rámci toho dochází nejprve k plánování potřebné týmové kapacity a dostupnosti. Dále jsou podrobně rozebírány jednotlivé funkcionality a činnosti, které mají být v rámci dané iterace splněny. Pokaždé se vybere činnost s nejvyšší prioritou[1], je provedena její důkladnější analýza a popis a na základě toho dojde k prvotnímu namodelování potenciálního řešení. Následně už probíhá

[1] Prioritu lze podle DAD určovat např. podle nejvyšší přidané hodnoty, rizika, operační nutnosti, zbývajícího času na dokončení nebo podle různých závislostí, vazeb a vztahů k jiným funkcionalitám.

dekompozice této činnosti do několika menších úkolů. Poté může v některých případech následovat přihlašování členů týmu o právě definované elementární úlohy. Nakonec dojde ke stanovení či aktualizaci časových odhadů. Pokud zbývá v rámci iterace ještě nějaký volný čas, je zpracována další činnost s nejvyšší prioritou. V opačném případě dojde k uzavření práce v rámci iterace.

DAD doporučuje po dokončení návrhu plánu iterace ještě jistou formu sekundární globální kontroly. Při té je znovu ověřena proveditelnost naplánované iterace, jelikož z tohoto pohledu mohou být zřejmé některé problémy (např. lze odhalit, jestli některý z členů týmu nemá přiděleno příliš práce). Poté bývá vhodné získat souhlas všech členů týmu s navrženým plánem. Nakonec dochází k tvorbě a sestavení plánu ve vizuální formě. DAD také doporučuje velmi hrubé zamyšlení nad plánem příští iterace.

Vývoj v rámci konstrukční iterace

Nejpodstatnější část práce na projektu probíhá právě v této části konstrukční fáze. Celý agilní tým spolupracuje na dodávce dílčího řešení v rámci iterace. Probíhají zde činnosti jako je programování, analýza, tvorba testů, integrace, oprava defektů, nasazování, meetingy atd.

Činnosti v rámci této části probíhají v trochu odlišném režimu, než je tomu u ostatních fází. Nejsou totiž prováděny sekvenčně, nýbrž se každý den opakují. DAD doporučuje každodenní plánování týmové práce pro daný den (denní meetingy inspirované Scrumem nebo Kanbanem, týdenní statusy apod.). Akceptována je ale i možnost nemít tyto koordinační schůze vůbec. Framework opět popisuje všechny tyto možností a také jejich výhody a nevýhody.

Na základě těchto meetingů pak probíhají práce, jejichž účelem je společně vyvinout hodnotné části řešení. Aby toho bylo dosaženo, DAD popisuje mnoho různých postupů a metod při řešení několika základních činností. Mezi ně patří např. lepší pochopení požadované části funkčnosti, její podrobné prozkoumání, nalezení řešení a také její vyvinutí. Dále pak popisuje různé postupy při ověření funkčnosti a při integraci. Také je doporučeno časté nasazování a sdílení funkcionality se stakeholdery.

V průběhu dne může vyvstat potřeba řešit některé problémy nahodile. Jedná se např. o tvorbu či úpravu dokumentace, řízení týmu, změnový a konfigurační management apod. DAD také hovoří o vývoji řízeném testy, kontinuální integraci a nasazování, nezávislých paralelních testech nebo o provádění kontrolních revizí. Také je popisována nutnost stabilizace každodenní odvedené práce.

Uzavření konstrukční iterace

V tuto chvíli by měl tým uzavřít a zhodnotit celou aktuální iteraci. Měl by být schopen předvedení funkcionality v rozsahu, který byl stanoven v rámci procesu jejího plánování. Také je to dobrý čas na zhodnocení stavu projektu, vyhodnocení výkonu týmu, identifikace možností ke zlepšení či diskuse se stakeholdery o tom, jak dále pokračovat.

DAD zde opět definuje mnoho různých možností a přístupů, které existují při demonstraci řešení před stakeholdery a při učení se z nabytých zkušeností z iterace (retrospektiva, dotazníky apod.). Dále je vyhodnocen aktuální postup, zbývající rizika a na základě toho je následně, v případě potřeby, upraven projektový plán nebo plán nasazení. Také dojde k nasazení aktuální verze na různá prostředí a je určena strategie pro následující období.

5.3.3 Fáze předání

Jedná se o finální fázi projektu. V rámci každé konstrukční iterace dochází k vytvoření použitelné části řešení. V této fázi je řešení již finální, stabilní a připravené k předání. Primární činností zde většinou bývají přípravy na nasazení produktu do produkce. Důležité je se ujistit, že řešení je pro produkci připraveno a že jsou na jeho převzetí připraveni i klíčoví stakeholdeři. V rámci této fáze může také docházet k opravě některých drobných a vizuálních defektů, doladění akceptačních testů a kritérií, tréninkům koncových stakeholderů, a případně mohou být také prováděny přípravy pro předání projektu servisnímu týmu.

Nejprve je nutné důkladné naplánování všech činností v rámci této fáze. Následně je třeba provést finální testy, připravit se na migraci dat a dokončit dokumentaci. Poté jsou o průběhu informování všichni zainteresovaní stakeholdeři, je proveden trénink určité podskupiny z nich a je zaškolen také podpůrný tým. Pokud je vše v pořádku, následuje proces samotného nasazení.

Po čas této fáze by nemělo docházet k zanedbávání příprav, k přijímání požadavků na novou funkcionalitu nebo k předání systému nepřipraveným uživatelům. Také není doporučeno přenášet zodpovědnost na servisní tým či příliš brzo rozpouštět celý stávající vývojový tým.

5.4 Případové studie

DAD na svých stránkách aktuálně zveřejňuje 6 případových studií, k nimž připojuje ještě další 4 krátká videa se stručným shrnutím procesu implementace frameworku. Většina projektů je z finančního a bankovního sektoru nebo z oblasti pojišťovnictví. V následující části se nachází stručný popis dvou úspěšných implementací DAD – pro finanční společnost Barclays a pro skupinu pekáren a kaváren Penera Bread. [21]

5.4.1 Barclays

Hlavním faktorem pro implementaci tohoto frameworku byl tlak na změnu projektového řízení vstříc agilnímu a štíhlému přístupu. V prvním roce transformace došlo k implementování agilního přístupu v rámci více než 800 týmů. Jedná se tak o jednu z největších implementací agilního frameworku. Rozhodnutí pro výběr DAD padlo především z důvodu, že tento framework nenabízí jeden striktně definovaný přístup, jenž by měl být dodržován napříč celou organizací. Místo toho poskytuje možnost volby a popisuje její klady, zápory a možné problémy.

Implementací frameworku DAD došlo k 300% nárůstu počtu dokončených funkcionalit za měsíc (z důvodu vyšší efektivity i díky členění na menší části). Více než polovina projektů produkuje novou verzi aplikace v intervalech kratších než 4 týdny. O 50 % vzrostl poměr kódu pokrytého testy. Na základě dotazníků pak vyplynulo, že spokojenost týmů je po implementaci agilního přístupu mnohem vyšší. Také mohou rychleji dodávat produkty na trh a pružněji reagovat na změny. [22]

5.4.2 Penera Bread

Velmi rychle rostoucí společnost Penera Bread má více než 1700 pekáren a kaváren v USA a Kanadě. Hlavní motivací pro implementaci frameworku byla potřeba zlepšení reakce na neustále se měnící podmínky v rychle rostoucím odvětví, které je navíc velice konkurenční. Vyvíjená řešení jsou mixem vývoje na zakázku, produktového vývoje a mobilních aplikací. Některé jsou vyvíjeny interně, jiné externě. Proto byl zvolen framework, který se zabývá různými typy vývoje software.

Adopce DAD dopomohla ke zvýšení frekvence dodávek vyvíjených produktů a ke zlepšení jejich kvality. Příznivě byly ovlivněny i pracovní vztahy mezi technickým a obchodním oddělením. Zlepšení transparentnosti v rámci všech týmů umožnilo redukci mnoha zbytečných meetingů, a také byl omezen počet nejrůznějších nedorozumění i dalších komunikačních problémů. [23]

6 LARGE-SCALE SCRUM

Framework LeSS (Large-Scale Scrum) je ve své podstatě stále klasický Scrum. Vychází ze všech principů a základních poznání standardního jedno-týmového Scrumu. Proto je nutné, aby organizace, která adoptuje a implementuje LeSS, plně rozuměla základním postupům a principům Scrumu. LeSS však je, stejně jako SAFe nebo DAD, framework pro rozsáhlé projekty a velké týmy.

Mezi hlavní zdroje informací, které byly v rámci této kapitoly využity, patří knižní publikace o škálování štíhlého a agilního přístupu [24], oficiální online znalostní báze frameworku [25] a případové studie [26]. V této kapitole se nachází konsolidovaný stručný popis metodiky LeSS.

Tento framework lze chápat jako jistou formu nadstavby Scrumu, která navíc obsahuje postupy pro organizování více týmů. To je věc, kterou Scrum opomíjel. Pro velké projekty vznikaly i další frameworky, které se Scrumem inspirovaly. Ty ale používaly i některé praktiky z jiných metodik a přístupů. LeSS však cílí výhradně na dodržování postupů známých z jedno-týmového Scrumu a na jejich rozšiřování. V žádném případě se tak nejedná o situaci, kdy je Scrum dodržován pouze na úrovni týmu, přičemž na vyšších úrovních již jeho myšlenky aplikovány nejsou.

Framework LeSS nabízí a popisuje 2 rozdílné typy implementace. První z nich je standardní LeSS (viz příloha C), který je vhodný do velikosti osmi týmů, z nichž každý čítá okolo osmi členů. Druhá varianta bývá označována jako LeSS Huge (velký LeSS). Ten může mít i několik tisíc členů na jednom projektu. LeSS Huge byl zaveden především proto, že takto velké týmy už nejsou spolehlivě řiditelné jediným Product Ownerem, což je základní princip Scrumu (viz příloha D). Produktový backlog je pro něj zkrátka příliš obsáhlý. Obě tyto varianty mají mnoho společných vlastností. Mezi ty klíčové patří:

- existence právě jednoho Product Ownera,
- jeden společný produktový backlog pro všechny týmy (přičemž backlog sprintu je specifický pro každý tým),
- shodně definovaný a časově ohraničený sprint,
- jeden společný, použitelný a plně integrovaný produktový přírůstek na konci každého sprintu.

Tato práce se nezabývá popisem charakteristik klasického Scrumu. Znalost jeho základních praktik a postupů je předpokládána. Mnoho z nich bylo navíc již částečně popsáno v rámci představení frameworků SAFe a DAD.

V následujících částech jsou popsány základní charakteristiky LeSS, hlavní rozdíly oproti klasickému Scrumu a struktura týmu. Následuje také popis LeSS Huge. V poslední části kapitoly jsou zmíněny 2 případové studie.

6.1 Standardní LeSS

Jak již bylo řečeno, existují dva typy frameworku LeSS pro různě rozsáhlé projekty. Pro jednodušší volbu mezi nimi byla definována přibližná velikost projektu (do velikosti zhruba osmi týmů by měl být používán standardní LeSS). Tato exaktní definice je však pouze orientační. Skutečný přelomový okamžik, mezi těmito dvěma verzemi, je ten, když jediný Product Owner již začíná ztrácet schopnost samostatně obsáhnout řízení celého projektu.

LeSS počítá s existencí pouze jediného Product Ownera, který je společný pro všechny týmy na projektu. Ten by se měl v první řadě snažit přenést maximum svých kompetencí na samotný tým. Některé zodpovědnosti mohou být také delegovány na vhodného experta v dané oblasti, se kterým úzce spolupracuje. Jedná se např. o přenesení následujících činností:

- rozdělování a definice dílčích funkčních požadavků,
- tvorba detailní analýzy,
- přímá komunikace se zákazníkem,
- činnosti zahrnující řízení projektu,
- řešení příliš podrobných detailů.

6.2 Rozdíly oproti Scrumu

Jelikož se klasický Scrum zabývá potřebami pouze jediného týmu, bylo zapotřebí v LeSS pokrýt i situace s více týmy. Proto došlo k úpravám některých známých Scrumových činností. Jiné aktivity byly nově zavedeny:

- Plánování sprintu – tato činnost se obvykle skládá ze dvou kroků:
  - První část – aktéry jsou Product Owner a členové týmů. V případě, že se projektu účastní takové množství týmů, že se všichni jejich členové pohodlně vejdou do společné místnosti, je preferována tato možnost. V opačném případě jsou vybráni zástupci každého z týmů (1–2 členové). Je nežádoucí, aby kterýkoli z těchto zástupců byl v roli Scrum Mastera. Při plánování pak probíhají klasické Scrumové činnosti, pouze je zde zahrnuto více týmů (např. dochází k nabízení a přihlašování se o jednotlivé položky z backlogu).
  - Druhá část – obvykle následuje bezprostředně po ukončení první části. Tato část probíhá již v rámci každého týmu odděleně. Všichni členové týmu si podle výsledků předchozí části sestavují vlastní backlog sprintu.
- Denní Scrum – jedná se o klasický každodenní meeting, který si řídí týmy samostatně. Je však doporučeno, aby měly jednotlivé týmy své meetingy naplánovány na rozdílné časy. V takovém případě pak mohou být vyslání jejich zástupci, aby

se zúčastnili meetingu jiného týmu. Tato účast by však měla být pouze pasivní.

- Údržba produktového backlogu – Scrum doporučuje provádět tuto činnost průběžně, přičemž by měla zabírat zhruba 5–10 % času sprintu. V kontextu rozsáhlých projektů je však lepší, pokud je pro tento účel vyhrazen specifický meeting (např. 4 hodiny pro každý sprint). Této schůzky by se pak měl opět účastnit Product Owner společně se členy týmu (případně s jejich vybranými zástupci).
- Zhodnocení sprintu – tato událost by měla probíhat v obdobné sestavě, jako tomu je v předchozím bodu. Navíc je v některých případech vhodná i účast Scrum Mastera, jelikož on je hlavní osobou řešící nastalé problémy v průběhu sprintu.
- Retrospektiva sprintu – každý tým by měl provádět svou vlastní retrospektivu. Měla by být prováděna na samém konci každého sprintu (např. v jeho posledním týdnu).
- Společná (projektová) retrospektiva – jedná se o volitelnou činnost, která umožňuje zhodnocení fungování projektu jako celku. Na rozdíl od ostatních činností je zde přítomnost Scrum Masterů přímo vyžadována, jelikož oni nejlépe dokáží vyhodnotit fungování týmu v rámci využívané metodiky. Tato událost je vždy plánována až po ukončení individuálních týmových retrospektiv. V případě potřeby ji lze provádět i v prvním týdnu následujícího sprintu.

6.3 Klíčové praktiky

Stejně jako tomu bylo u ostatních metodik, i LeSS definuje a popisuje sadu praktik, principů a charakteristik, na kterých je postaven. Všechny tyto vlastnosti jsou společné jak pro standardní LeSS, tak také pro LeSS Huge. Některé z nich jsou zcela totožné se zásadami Scrumu, jiné byly přidány pro potřeby rozsáhlých projektů a pro zlepšení procesu řízení projektu. Díky nim lze získat lepší představu o tom, jak LeSS funguje a jaké hodnoty vyznává.

Spousta z těchto principů se vzájemně prolíná, odkazují na sebe a doplňují se. Díky tomu působí framework velmi jednotným a skutečně promyšleným dojmem.

6.3.1 LeSS je Scrum

Tento bod byl již popsán v předchozích odstavcích. LeSS skutečně není novou formou Scrumu a není ani žádnou jeho vylepšenou verzí. Jedná se o soubor pravidel a tipů, které jsou ověřené a fungují pro distribuované týmy pracující na rozsáhlých projektech.

LeSS je framework, který odhaluje procesní problémy organizace. Scrum se totiž soustředí pouze na malé týmy a malé skupiny lidí, a proto dokáže efektivně odhalit pouze lokální a dílčí problémy. Jelikož je LeSS zaměřen na podstatně větší skupiny lidí, snadněji

vyplynou na povrch nedokonalosti v rámci organizační struktury, procesů, systému odměn, lidí nebo úkolů. Pokud jsou jeho zásady důsledně dodržovány, každý z týmů by se měl postupně s každým sprintem zlepšovat.

6.3.2 Méně je více

Tato zásada vychází z poznání, že nevýhody plynoucí ze zavedení komplexního organizačního řešení jsou mnohdy závažnější než všechny problémy, které se toto komplexní řešení pokouší napravit. Cílem by tedy mělo být spíše hlubší porozumění podstatě těchto problémů a jejich vyřešení pomocí jednoduchých řešení.

LeSS se pouze snaží definovat ta nejnutnější opatření nutná pro vývoj rozsáhlých projektů. Framework tak zcela jednoznačně odmítá přidávání dalších rolí, artefaktů a procesů, jelikož jsou brány v potaz nevýhody a problémy související s těmito přídavky:

- Přidávání více rolí vede k méně zodpovědnému týmu.
- Více přidaných procesů vede k omezení zodpovědnosti mezi týmy. Také to není v souladu s principy kontinuálního zlepšování, jelikož lidé pak pouze následují proces bez toho, aby jej chápali, učili se jej a zlepšovali ho.
- Přidávání dalších artefaktů odvádí tým od soustředění se na zákazníka.

6.3.3 Štíhlé myšlení

Princip štíhlého myšlení (*Lean thinking*) je jedním z klíčových přístupů uplatňovaných u rozsáhlých projektů. Původně se jednalo o systém praktik využívaných ve společnosti Toyota. Je to přístup, který se prolíná téměř všemi činnosti a úrovněmi projektového řízení v IT. To zahrnuje např. vývoj produktu, prodej, údržbu nebo správu lidských zdrojů. [25]

Celý tento přístup je rozdělen do šesti částí:

- Cíle štíhlého myšlení – rychlé dodání produktu v nejlepší možné kvalitě a s co nejvyšší hodnotou pro zákazníka či společnost. Zákazník musí být výsledkem nadšen, přičemž náklady společnosti musí být co nejnižší. Ve společnosti musí panovat dobrá morálka a musí být dodržovány nejvyšší bezpečnostní standardy.
- Základy štíhlého myšlení – vrcholný management musí plně rozumět těmto principům, musí je dokázat efektivně aplikovat, a především je musí umět správně předávat a učit další články v projektu i organizaci. Všechna rozhodnutí musí být činěna s ohledem na dlouhodobou filozofii společnosti.
- Kontinuální zlepšování – tento rozsáhlý princip je z pohledu LeSS natolik zásadní, že jej pojal za svůj a je tak popsán ve speciální kapitole 6.3.7.
- Respektování lidí – zde je samozřejmě cílem projevovat respekt k lidem v projektu a dodržovat určitá pravidla morálky. Mimo to je zde však také kladen důraz na to,

aby lidé nedělali zbytečnou práci, fungovali jako tým a dodržovali pravidla týmové práce. Perspektivní lidé by měli být správně vedeni a mentorováni. Všichni musí pracovat v čistém, příjemném a bezpečném prostředí. Také je zde zahrnut důraz na samořídící se týmy, které obsahují veškeré potřebné dovednosti a jsou přímo spojeny se zákazníkem (jsou co nejblíže byznysu). S tím také souvisí existence respektu vůči zákazníkovi.

- Skupina tzv. 14 principů – jedná se o soubor praktik které jsou v souladu se štíhlým myšlením. Jsou to např. pravidla a postupy pro zajištění integrované kvality, vizuální management, výchova expertů a leaderů, používání ověřených technologií, činění rozhodnutí na základě koncensu a další.
- Štíhlý vývoj produktu – mezi praktiky v této části patří např. správně zvolená kadence, časové ohraničování, opětovná použitelnost informací a znalostí, paralelní tvorba více alternativních designů, zavedení týmové místnosti s vizuálním managementem aj.

6.3.4 Systémové myšlení

Tento princip byl již zmíněn a částečně rozebrán v rámci popisu frameworku SAFe (viz kap. 4.2.2). Projektoví manažeři mívají v rámci snahy o optimalizaci systémových procesů několik obvyklých problémů. Mezi nejčastější z nich patří nízká znalost o systémové dynamice, vyhodnocování zpětných vazeb nebo nedostatečné porozumění základním příčinám problémů a jejich nalézání. Problémem také je, že po lokálním vyřešení problému nevědí, zda toto řešení zabralo nebo jak a proč ovlivnilo celkový výkon systému (ať už pozitivně, či negativně).

Systémové myšlení pomáhá řešit lokální problémy s využitím globálních poznatků a souvislostí. Na vše je potřeba nahlížet z vyšší perspektivy a uvědomovat si širší vztahy mezi jednotlivými procesy a činnostmi. Jedná se o sadu nástrojů, které pomáhají odhalit a pochopit systémovou dynamiku, vizualizovat mentální modely, a nakonec nalézt a navrhnout lokální optimalizaci. [25]

6.3.5 Empirická procesní kontrola

Toto je jeden ze zcela stěžejních principů Scrumu, tím pádem i frameworku LeSS. Scrum by totiž neměl být chápán jako proces nebo technika pro vývoj produktu. Jedná se spíše o jistý rámec, který tyto procesy a techniky využívá, a díky kterým lze postupně dosahovat zlepšení v oblastech vývoje.

Mnoho jiných metodik se snaží vývoj software popisovat příliš detailně. Tým pak má jen velmi omezené možnosti pohybu v rámci mezí dané metodiky. To není příliš žádoucí.

Z toho důvodu LeSS pouze nastiňuje možná řešení, myšlenky a přístupy, což poskytuje větší manévrovací prostor. Záměrně jsou proto některé praktiky popisovány pouze velmi obecně. Je pak na projektovém managementu, aby si je upravil a doladil podle svých aktuálních potřeb.

V rámci tohoto principu jsou využívány krátké vývojové cykly, ve kterých je vytvořena malá použitelná část produktu. Následně dochází k důkladnému prozkoumaní toho, co bylo skutečně vytvořeno a jakým způsobem. Díky tomu může dojít k přizpůsobení vyvíjeného produktu a také samotného procesu jeho vývoje. V rámci vývoje by také měly být zakomponovány mechanismy zajišťující transparentnost, díky níž pak může docházet k procesu inspekce.

6.3.6 Transparentnost

Bez zajištění transparentnosti je velmi obtížné projekt řídit a přizpůsobovat se aktuálním potřebám a identifikovaným problémům. Přesto je však tento princip v mnoha případech záměrně ignorován. Realita totiž občas odhaluje velmi nepříjemná zjištění, která vedou k tvorbě jistého psychologického bloku. Velké množství problémů, souvisejících s lidmi, skupinami, procesy či nástroji, pak zůstává neodhaleno a nelze je včas a efektivně řešit.

Proto LeSS důrazně vybízí k adoptování tohoto principu. Zajištění transparentnosti a korektního sdílení poznatků a informací je vskutku stěžejní. Konkrétních činnosti, které k tomu dopomáhají, jsou např. redukce aktuálně probíhajících činností, omezení velikosti dávek nebo zkrácení doby trvání jednotlivých cyklů. Také je vhodné korektně definovat a evidovat klíčové činnosti v projektu (případně také ty, jenž nebyly dokončeny podle plánu).

6.3.7 Kontinuální zlepšování vstříc dokonalosti

Tento princip je jedním z pilířů štíhlého myšlení. V rámci LeSS není žádoucí existence jakékoli skupiny zabývající se změnami či jinou formou změnového managementu. Vše by mělo probíhat průběžně. Jedná se o proces, který je v zásadě nikdy nekončící.

Cílem je dosažení dokonalosti, kterou však nelze exaktně definovat, takže pro každý projekt či organizaci může být její definice rozdílná. Může se jednat například o dokončení projektu v dokonalé kvalitě, dosažení maximální spokojenosti klienta či předání projektu s velkým profitem. Může se také jednat o dokončení funkcionality s minimem defektů nebo s nízkým úsilím a náklady. Dokonalost může být hodnocena i v kontextu spokojenosti zaměstnanců a jejich neustále se zvyšujících zkušeností a dovedností.

V rámci tohoto principu LeSS popisuje několik praktik a myšlenek. Jednou z nich je doporučení, aby management trávil více času společně s týmem a pochopil tak jeho pro-

blémy a potřeby. Na základě této zkušenosti lze pak činit kvalifikovanější rozhodnutí, která jsou postavena na základě širšího koncensu. Další myšlenkou je to, že by mělo docházet k neustálému zlepšování pracovních postupů a činností (tzv. *kaizen*). Základem je porozumění některé činnosti či procesu a následně její zlepšování na základě experimentů. Dále se pak jedná o neustálý tlak na zvyšování hodnoty, eliminaci nepodstatných či problémových činností, snahu o dokonalost, nastavení plynulého pracovního procesu bez zbytečných prostojů a o respektování lidí.

6.3.8 Zaměření na zákazníka

V rozsáhlých projektech bývá problémem udržet si dostatečně detailní povědomí o klientovi. Je důležité mít neustále na paměti, kdo zákazník je a jak bude vyvíjený produkt používat. Udržování kontaktu a spojení s klientem je tak naprosto stěžejní.

LeSS definuje několik praktik, jak naplnění tohoto principu dosáhnout. Tým by měl vytvářet komplexní a ucelenou část funkcionality, nikoli pouze dílčí komponenty. Každá skupina týmů by se měla věnovat určité oblasti. Product Owner by měl fungovat spíše jako spojovací článek mezi týmem a zákazníkem, nikoli jako pouhý zprostředkovatel informací. Je důležité, aby konstantně docházelo k vyhodnocování priorit položek v produktovém backlogu. Týmy by také měly preferovat možnost konzultací a analýzy přímo s klientem (raději než s Product Ownerem).

6.3.9 Vnímání produktu jako celku

Je důležité mít na paměti, že zákazník skutečně platí za celý produkt a nikoli za jeho dílčí části. Ty části, které zatím nejsou plně integrovány do zbytku projektu, mu nepřinášejí žádnou přidanou hodnotu. Tým, který dokončí jistou část projektu, není se svou prací hotov, dokud tato část není plně integrována s prací ostatních týmů. Kdykoli se naskytne možnost volby mezi optimalizací výstupu jednoho z týmů a celkovým projektem, vždy cílíme na výkon systému jako celku.

Dosažení takovéhoto nadhledu u každého z týmů bývá často velmi obtížné. Je však nezbytné, aby si týmy tuto zásadu osvojily, nevnímaly se jako konkurenci a spolupracovaly při plnění společných cílů.

6.3.10 Teorie front

Princip teorie front nabízí vhled do problematiky, proč bývá vývojový proces zbytečně pomalý. To platí speciálně pro velké projekty. LeSS tak v rámci tohoto principu nabízí několik typů na zlepšení tohoto problému.

Správné řízení front umožňuje redukci celkové doby trvání cyklu. V rámci vývoje software se fronty nacházejí na mnoha různých místech. Jedná se např. o projekty a produkty ve firemním portfoliu, tvorbu nových funkcionalit v rámci produktu, o návrhy specifikací a dokumentací nebo o funkcionality čekající na otestování či integraci.

V rámci tohoto přístupu je preferována možnost maximální eliminace front (spíše než jejich management). To znamená, že se má pracovat na řešení příčiny jejich tvorby a nemá docházet k řešení již nastalého problému. To zahrnuje např. změnu systému organizace práce, vývoje, nástrojů, procesů, praktik, předpisů apod.

V některých případech však vznik front systémově eliminovat nelze nebo tyto fronty již existují. V takových případech tak samozřejmě musí dojít k jejich řízení. Toho lze dosáhnout pomocí omezení velikosti jednotlivých dávek (přičemž všechny by měly být rozsahově podobné), redukcí množství aktuálně prováděné práce nebo zmenšením velikosti fronty. Poté je již lze kontrolovat a řídit pomocí vizuálního managementu. [25]

6.4 Struktura týmu

Ačkoli se framework LeSS zabývá výhradně rozsáhlými projekty, jednotlivé týmy zůstávají i tak základní stavební jednotkou každého projektu. Složení každého z těchto týmů je téměř totožné, jako tomu bývá u standardního Scrumu. Měl by se skládat ze Scrum Mastera a jednotlivých členů, kteří pokryjí všechny potřebné role a činnosti. Product Owner je pro daný projekt pouze jeden a je tedy společný pro všechny týmy.

Každý tým by měl sdílet zpracovávaný úsek práce. Práce každého člena pak musí být provázána i s vykonanou činností ostatních – je zde tedy vzájemná závislost. Tým společně sdílí zodpovědnost za přidělené úkoly, ke kterým se všichni zavázali, a pracuje na základě sady společných pracovních dohod a postupů.

Každý člen musí být součástí pouze jednoho týmu, čímž je dosažena maximální míra oddanosti týmu a 100% soustředění na potřeby a cíle daného týmu. Je velmi důležité, aby každý tým obsahoval všechny potřebné role a mohl tak samostatně dokončit odsouhlasenou část práce. Všichni členové týmu by měli pracovat společně a být v co nejčastějším kontaktu. Ideálním stavem je, pokud všichni sdílí společnou místnost. Tým by také měl držet pohromadě co nejdelší možnou dobu. Jednotliví členové se tak lépe poznají a postupně se naučí, jak optimalizovat svou práci jako celek.

Důraz je kladen na to, aby týmy byly skutečně samořídící. Všechna rozhodnutí by měla být činěna po interní diskuzi a na základě společného koncensu. Velmi důležitým bodem je také to, že si každý tým řídí své externí závislosti svépomocí. To platí zejména v kontextu rozsáhlých projektů. Jednotlivé týmy jsou samy zodpovědné za koordinaci práce mezi nimi a ostatními týmy. Neměla by tedy existovat žádná vyšší ani centrální autorita, která

by toto jakkoli řídila za ně. Tento bod také přispívá ke kvalitnějšímu procesu kontinuální integrace. Tým není se svou prací hotov, dokud daná část není plně integrována se zbytkem vyvíjeného projektu. Nelze se tedy spoléhat na žádnou třetí stranu, která by integraci zajistila.

6.4.1 Role managementu

LeSS má velký dopad i na řídící struktury společnosti, především pak na střední management. Ten zde má odlišnou roli a postavení, než tomu bylo u tradičních metodik. Jelikož by jednotlivé týmy měly být maximálně samořídící, odpadá zde nutnost pro management tyto týmy jakkoli dále řídit. Z manažerů se tak stávají spíše kouči a mentoři, kteří dohlíží na dodržování principů, zásad a globálního směřování projektu a společnosti.

V tradičních metodikách manažeři poskytovali odpovědi na 2 základní otázky: co dělat a jakým způsobem danou činnost provést. Tyto kompetence jsou však v LeSS přeneseny na jiné články týmu. Rozhodování o tom, co se má dělat, bylo přesunuto na Product Ownera. Ten má totiž mnohem lepší přehled zevnitř o tom, co má v danou chvíli největší prioritu, co přináší nejvyšší přidanou hodnotu pro zákazníka a jaká činnost je nerizikovější. Rozhodování o samotném provedení daného úkolu pak spadá, jak již bylo zmíněno, plně do kompetencí jednotlivých samořídících týmů.

Manažeři se tak mohou soustředit na vnímání projektu jako celku, případně na projekt v kontextu celé organizace. Měli by pomoci Scrum Masterům i jednotlivým týmům k odstranění všech nastalých překážek a nepříjemností. Jejich úkolem je naučit členy týmu, jak samostatně řešit problémy a podporovat je ve společném a systematickém zlepšování.

6.5 LeSS Huge

Tuto verzi frameworku lze chápat jako rozšíření či nadstavbu standardního LeSS pro ještě rozsáhlejší projekty (např. nad velikost osmi týmů). Nejedná se tedy o framework, který by standardní LeSS nahrazoval, nýbrž jej pouze doplňuje.

Pokud si Product Owner ani s pomocí týmů již nedokáže zachovat dostatečný přehled o projektu a produktovém backlogu, je nutné rozdělit daný projekt do menších logických částí – tzv. oblastí požadavků. Jeho kompetence jsou pak částečně delegovány mezi speciální členy týmu. Pro tyto potřeby je tak definován jeden nový artefakt a dvě speciální role:

- Oblastní backlog – jedná se o artefakt, který tvoří separátní backlog, jenž pokrývá určitou část požadavků klienta. Je vytvořen na základě produktového backlogu a nabízí tak vhled do jeho konkrétní části. Tato část zpravidla bývá vnitřně logicky spjata a provázána (obsahuje související funkce).

- Oblastní vlastník produktu (*Area Product Owner*) – nově přidaný člen týmu, na kterého byla přenesena část kompetencí Product Ownera. V podstatě jej lze chápat jako klasického Product Ownera, který má však zodpovědnost pouze za jemu přidělenou oblast projektu (za jeden oblastní backlog). Je doporučeno, aby v jeho kompetenci bylo 5–8 týmů.
- Tým Product Ownera – tento speciální tým je tvořen všemi Product Owners – oblastními i tím hlavním. Společně se podílejí na rozhodování ohledně stanovení rozsahu požadavků, provádějí prioritizace a plánují nadcházející činnosti. Finální slovo však stále závisí na centrálním Product Ownerovi, který ostatní členy řídí a kontroluje.

V rámci činností jsou v LeSS Huge také jisté drobné změny:

- Předběžné plánování sprintu – předchází standardnímu plánování sprintu (zpravidla je vykonáváno v posledním týdnu sprintu předchozího). Tým Product Ownera společně koordinuje celkovou prioritizaci v rámci produktového backlogu. Jsou zde probírány priority jednotlivých oblastních backlogů a probíhá diskuze o aktuálním stavu v rámci každé oblasti. Tým by se měl takto setkávat každý sprint, aby bylo vyhodnoceno, v jaké fázi se celý projekt nachází a kam by měl dále postupovat.
- Plánování sprintu – probíhá pro každou oblast odděleně. Místo hlavního Product Ownera zde však zastupuje ten oblastní. Účastní se jej pouze týmy z dané oblasti.
- Údržba produktového backlogu – je prováděna v rámci dané oblasti. Účastníky jsou oblastní Product Owner a oblastní týmy (případně pouze jejich zástupci).
- Zhodnocení sprintu – i zde jsou zastoupeni pouze členové dané oblasti. Může se jí však účastnit i hlavní Product Owner, pokud jej něco konkrétního zajímá nebo pokud má nějaké dotazy či postřehy.
- Společné zhodnocení – jedná se o volitelnou aktivitu, která je vykonána pouze v případě potřeby. Účastní se jí celý tým Product Ownera a zástupci jednotlivých agilních týmů. Jsou zde diskutovány funkcionality, možná vylepšení a problémy, které zajímají tým Product Ownera. Také zde může dojít k předvedení některých funkcionalit, které jsou pro projekt klíčové nebo jsou důležité pro nadcházející sprinty.
- Společná retrospektiva – je to také volitelná činnost, které se účastní reprezentanti jednotlivých týmů. Je prováděna v případě potřeby zlepšení učení a postupů v kontextu celého systému (může se dotknout i oblasti firemní kultury apod.). Může být prováděna na úrovni dané oblasti i na úrovni celého produktu.

6.6 Případové studie

Na stránkách frameworku LeSS se nachází téměř třicet různých případových studií. Spousta z nich je z oblasti telekomunikačních systémů a technologií, další jsou z oblasti bankovnic-

tví, pojišťovnictví, výroby aj. V další části jsou popsány studie automobilové společnosti BMW Group a telekomunikační firmy Nokia. [26]

6.6.1 BMW Group

Hlavním projektovým cílem byl vývoj nových systémů pro společnost BMW, které měly sloužit pro podporu prodejního procesu. Ty však měly být navíc plně integrovány do již existujících více než 80 dalších systémů. LeSS poskytl jasné vedení v průběhu celého projektu a pomohl i v několika kritických projektových rozhodnutích.

Framework dopomohl k přechodu vstříc mnoha kooperujícím týmům, které fungují plně v souladu s agilními principy. Bylo umožněno nahlížet na celý rozsáhlý projekt jako na celek a došlo k jasné definici cílů, které měly být splněny v rámci integrace s každým externím systémem. Projekt byl po celou dobu přehledný a plně transparentní. Klient měl dostupnou plně funkční a verzi po každém sprintu, díky čemuž bylo možné pružně reagovat na změnové požadavky, čímž byla zajištěna jeho spokojenost. Empirický proces umožnil poskytování realistických odhadů a byla tak výrazně zlepšena celková předvídatelnost projektu. [27]

6.6.2 Nokia Networks

Projekt se zabýval zejména výstavbou nové vysokokapacitní sítě. Hlavním rizikem bylo použití zcela nové technologie, která byla neznámá a nikdy předtím nebyla v rámci organizace využita. Druhým problémem byla nejasná a neúplná definice zadání. To se zpřesňovalo až s průběhem projektu (na základě postupného učení). Implementace frameworku LeSS se zdála být vhodnou odpovědí na tyto zásadní otázky.

Díky dodržování zásad frameworku a agilního přístupu došlo ke znatelnému zrychlení času dodávky produktu na trh. Navíc byla poskytnuta velká míra flexibility. Oproti sekvenčním přístupům trvala délka projektu pouze polovinu času, a i tak byla umožněna reakce na velké množství zásadních změn, které by při sekvenčním vývoji nebyly možné. Díky automatizovaným akceptačním testům bylo snadné zachovat stávající kvalitu kódu při přidávání nových týmů. [28]

7 DETEKCE AGILNÍCH PRAKTIK

Jedním z cílů této práce je definice sady praktik, které jsou charakteristické pro agilní vývoj. Jelikož jsou popisované frameworky určené pro škálování agilního přístupu, bude v rámci popisu jednotlivých praktik kladen speciální důraz na jejich dodržování v rozsáhlých projektech.

V rámci této kapitoly je nejprve popsán experimentální nástroj SPADe, díky kterému lze praktiky vyhledávat. Součástí jeho popisu je také rozbor využívaného doménového modelu. Poté následuje samotný popis jednotlivých detekovatelných praktik a definice konkrétních symptomů, pomocí kterých lze tyto praktiky v reálných datech nalézt.

7.1 Nástroj SPADe

V rámci Katedry informatiky a výpočetní techniky (KIV) Fakulty aplikovaných věd (FAV) na Západočeské univerzitě v Plzni (ZČU) vzniká experimentální nástroj zvaný SPADe (*Software Process Anti-pattern Detector*). Tento software se zabývá především dolováním dat a informací o nejrůznějších projektových praktikách z tzv. *Application Lifecycle Management* (ALM) nástrojů. Zjištěné údaje jsou pak pomocí datové pumpy přenášeny a ukládány do jednotného datového skladu (viz obr. 7.1). [30]

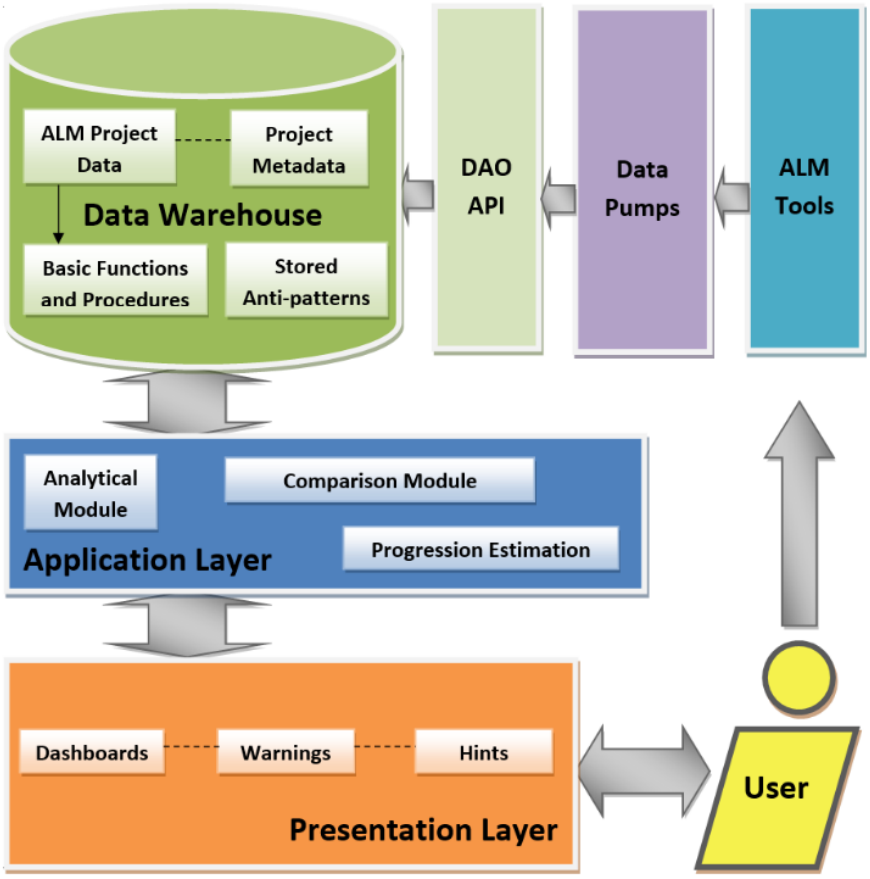
Na základě zjištěných údajů lze pak provádět rozličné analytické činnosti. Je možné např. detekovat často se opakující špatné praktiky (tzv. *anti-patterns*) projektového řízení. Informace o takto nalezených praktikách je pak předána uživateli (např. projektovému manažerovi nebo vedoucímu vývojového týmu), který na základě tohoto zjištění může včas vhodně přizpůsobit a pozměnit odpovídající projektové postupy a činnosti. Pomocí tohoto nástroje tak lze některé problémy, které by jinak zůstaly dlouho skryty, včasně a jednoduše odhalit.

Stejně tak existuje také možnost odhalování a detekce některých pozitivních praktik (tzv. *patterns*). Většina metodik má své charakteristické vzory a postupy. Spousta z nich je pak společná pro metodiky a frameworky vyznávající agilní přístup řízení vývoje software. Toho je využito i v kontextu této práce.

Jak již bylo zmíněno, SPADe využívá data získaná ze sady ALM nástrojů. To jsou podpůrné nástroje pro řízení životního cyklu aplikace. [30] SPADe čerpá data z následující základní sady nástrojů:

- SVN,
- Git,
- GitHub,
- Bugzilla,

- Redmine,
- Jira,
- Assembla.



Obr. 7.1: Celkový architektonický koncept nástroje SPADe [30]

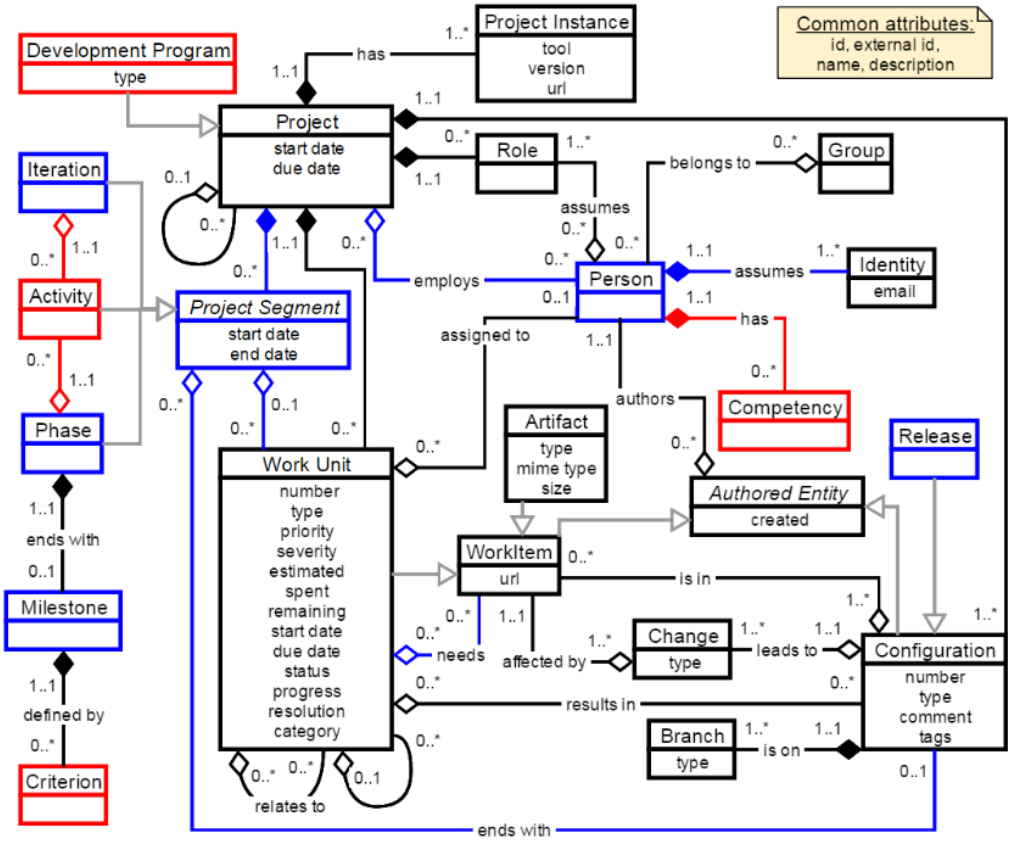
7.1.1 Datový model

Jelikož SPADe čerpá a ukládá data z různých ALM systémů, bylo nutné vytvořit jednotný datový model, ve kterém budou zjištěné údaje uchovávány. Data jsou tak ukládána v ucelené datové struktuře bez ohledu na to, odkud byla získána.

Doménový model nástroje SPADe (viz obr. 7.2) lze rozdělit do několika vzájemně provázaných částí. V první řadě jsou nadefinovány 4 základní koncepty, které definují proces vývoje každého projektu [30]:

- *Project* – obsahuje základní informace o projektu, které reprezentují data získaná ze všech dostupných zdrojů (z různých ALM nástrojů).
- *Work Unit* – jedná se o elementární úkol či ticket z nástrojů pro správu a sledování požadavků.

- *Role* – sada oprávnění a zodpovědností, které mohou být přiděleny členům týmu.
- *Artifact* – jedná se o výsledky nejrůznějších činností, které byly vyprodukovány v průběhu projektu.



Obr. 7.2: Doménový model nástroje SPADe [30]

Dále následují entity, které popisují různé projektové fáze a jejich cíle:

- *Iteration* – opakovatelný proces, který rozděluje projekt na menší části. Vychází z konceptu iterativních a agilních metodik.
- *Phase* – segment projektového životního cyklu, který vychází z konceptu sekvenčních metodik (Vodopádový model, V-Model) a UP.
- *Milestone* – u projektové fáze může být definována sada cílů, které je potřeba před jejím ukončením naplnit.
- *Criterion* – částečný cíl fáze, indikátor dosažení stanoveného milníku.

Podstatnou částí modelu jsou také třídy, které reprezentují osoby účastnící se prací na projektu:

- *Person* – umožňuje konsolidované sledování činností jednotlivých osob v rámci různých nástrojů.
- *Identity* – reprezentuje uživatelský účet v konkrétním ALM nástroji.

- *Group* – reprezentace týmu, organizace nebo jiné podskupiny lidí, kteří se podílejí na projektu.
- *Competency* – reprezentuje odbornost, certifikace či vzdělání, které může osoba mít.

Poslední skupinou jsou entity, jenž pomáhají k podrobnému technickému popisu samotného projektu:

- *Work Item* – seskupuje dohromady entity *Artifact* a *Work Unit* (tedy veškeré výstupy projektu).
- *Activity* – skupina vzájemně provázaných dílčích úkolů, které mají společný cíl a mohou reprezentovat různé praktiky.
- *Change* – reprezentuje změnu jedné položky *Work Item* v daném čase (přidání, modifikace nebo odstranění artefaktu).
- *Configuration* – stav všech položek projektu, který vyplynul ze sady provedených změn v daném časovém okamžiku.
- *Release* – stabilizovaná verze, která byla určena a označena pro nasazení.
- *Branch* – vývojová větev projektu.
- *Project Instance* – zachycuje informace o jednotlivých instalacích ALM nástrojů, ze kterých jsou sbírána data o projektu.
- *Development Program* – skupina souvisejících projektů (např. verze pro různé platformy).

7.2 Výběr detekovatelných praktik

Na základě doménového modelu nástroje SPADe je možné nalézt a definovat nejrůznější praktiky, postupy a činnosti, které jsou charakteristické pro agilní metodiky. Zároveň také existuje možnost hledání společných charakteristik pro všechny tři popisované frameworky, případně lze definovat i některé společné rysy pro metodiky, které jsou určeny speciálně pro rozsáhlé projekty.

Praktiky byly vybrány tak, aby odpovídaly moderním agilním postupům. Zvláštní důraz byl také kladen na jejich použití na rozsáhlých projektech. Vztahují se především k fungování týmů a k dodržování jistých postupů nebo činností.

Následující část se věnuje popisu charakteristik jednotlivých praktik. U každé z nich je také zaznamenán výčet jejích zachytitelných symptomů.

7.2.1 Aktivní komunikace mezi jednotlivými týmy

Je důležité, aby byla komunikace mezi jednotlivými týmy řízena jejich samotnými členy. Neměla by existovat jedna dedikovaná osoba, přes kterou by informace centrálně proudily. Každý by se měl na komunikaci aktivně podílet a řešit problémy samostatně. Jedná

se o jednu z klíčových charakteristik agilních týmů v kontextu rozsáhlých projektů. Je však velmi komplikované takovouto komunikaci zachytit, jelikož týmy mohou používat mnoho různých nástrojů (email, telefon, osobní setkání v kanceláři, Skype[1], Slack[2] apod.).

Symptomy

- Existuje komunikace mezi členy jednotlivých týmů.
- Komunikace není centralizovaná – neprobíhá pouze ze strany Scrum Mastera, Product Ownera nebo jiného dedikovaného člena.
- Komunikace se účastní více členů týmu (ideálně všichni).
- Mohou existovat koordinační meetingy, kterých se účastní pouze podmnožina týmů (např. pokud funkcionalita jednoho týmu zasahuje do prací jiného týmu).

7.2.2 Délka iterace

Společnou charakteristikou všech popisovaných frameworků jsou krátké a jasně časově vymezené iterace, díky nimž je projekt mnohem lépe řiditelný a plánovatelný. Také je zajištěno, že každý přírůstek bude generovat obdobnou přidanou hodnotu s konstantní kadencí.

SAFe doporučuje nastavit ideální délku iterace na 2 týdny. Připouští však i rozmezí mezi 1 až 4 týdny. Stejně je na tom i LeSS, který taktéž vychází ze zásad Scrumu. Naproti tomu DAD nabízí mnoho různých přístupů, které se liší v závislosti na kontextu projektu. DAD tak popisuje i extrémní varianty, jako jsou iterace přesahující 6 týdnů i variantu kompletně bez jakýchkoli iterací. Nicméně doporučený rozsah jsou méně než 3 týdny pro krátké iterace a 3–6 týdnů pro iterace středně dlouhé.

Symptomy

- Délka každé iterace by se měla pohybovat mezi 1–3 týdny.
- Všechny iterace by měly mít svou délku stanovenou stabilně – měly by být přibližně stejně dlouhé.

Možné vylepšení

- Každá iterace v rámci konstrukční fáze by se měla pohybovat mezi 1–4 týdny.
- Každá iterace v rámci ostatních fází by měla být maximálně 2 týdny dlouhá.
- Všechny iterace v rámci konstrukčních fází by měly mít svou délku stanovenou stabilně – měly by být přibližně stejně dlouhé.

[1] https://www.skype.com/
[2] https://slack.com/

7.2.3 Eliminace rizik v počátečních fázích

V úvodních fázích projektu i jednotlivých iterací by měl být kladen důraz na minimalizaci rizika. Je tedy vhodné zabývat se i takovými činnostmi, které nemají nejvyšší prioritu, avšak jsou poměrně hodně rizikové.

Symptomy

- Na počátku projektu jsou činnosti prioritizovány podle závažnosti (*severity*), nikoli pouze podle priority.
- Na počátku každé iterace jsou činnosti prioritizovány podle závažnosti (*severity*), nikoli jen podle priority.

7.2.4 Kolektivní vlastnictví projektu

Tento bod nezahrnuje pouze kolektivní vlastnictví programového kódu, nýbrž taktéž sdílené vlastnictví celého zbytku projektu (včetně dokumentací i dalších artefaktů). V projektech by v rámci jednotlivých agilních týmů měla panovat maximální důvěra. Může existovat jedna osoba, která by za daný prvek mohla mít primární zodpovědnost, avšak k modifikacím daného artefaktu by měl mít možnost volně přistoupit kterýkoli člen týmu.

Symptomy

- Dokumenty a aplikační designy jsou průběžně doplňovány a upravovány různými osobami.
- Části kódu související se stejnou funkcionalitou, jsou upravovány a měněny různými osobami.
- Nahlášené defekty opravuje libovolná osoba, nikoli pouze programátor, který danou funkcionalitu původně implementoval.
- Refactoring libovolné části kódu může provádět kterýkoli člen týmu.

7.2.5 Krátká inicializační fáze

Úvodní fáze projektu by neměly být příliš dlouhé. V jejich rámci by měla být provedena pouze obecná specifikace klientových požadavků a nástin rozvržení projektu. Detailnější specifikace pak mohou být prováděny již v rámci konstrukční fáze. Díky tomu může začít samotný proces vývoje software mnohem dříve, než pokud by se čekalo na kompletní dokumentaci. Cílem tedy je provádět v inicializačních fázích pouze nezbytné činnosti. Zbytek by pak měl být přesunut do následujících fází.

Symptomy

- Počáteční fáze projektu je ideálně 2 týdny dlouhá, maximální doba je pak jeden měsíc.
- Tato fáze bývá ukončena milníkem LCO (*Life Cycle Objectives*).

7.2.6 Multifunkční členové týmu

V rámci agilních týmů je vhodné, aby jednotliví členové týmu byli schopni zastat více různých funkcí a činností. Např. analytik se v jistých částech projektu může podílet na testování, reportování defektů či retestech. Tento bod výrazně usnadňuje komunikaci v rámci týmu a také pomáhá odstraňovat hierarchické struktury, což je zcela zásadní pro agilní metodiky.

Symptomy

- Člověk s rolí analytika nebo vývojáře (viz praktika 7.2.7) se občas účastní i hlášení či uzavírání defektů.
- Člověk s rolí vývojáře nebo testera v některých fázích projektu aktivně přispívá k úpravám dokumentací nebo vytváří detailní zadání pro vývoj.
- Člověk s rolí analytika nebo testera je schopen aktivního vývoje software – vypracovává jednoduchá vývojová zadání či opravuje nahlášené defekty.

7.2.7 Multifunkční týmy

Jak již bylo mnohokrát řečeno, agilní týmy by měly být poměrně malé. Kvůli tomu jsou kladeny vysoké nároky na to, aby přesto mohly pracovat maximálně samostatně. Proto by jejich členové měli být schopni obstarat veškeré potřebné činnosti. V každém týmu by tak měla být zastoupena alespoň jedna analytická, vývojářská a testerská role.

Symptomy

- Každý tým musí mít analytika – člověka, který vytváří a upřesňuje podrobná zadání pro vývoj (vytváří vývojové tasky), sepisuje a aktualizuje dokumentace, aplikační designy apod.
- Každý tým musí mít vývojáře – je to člověk, jehož primárním úkolem je vypracovávání vývojových úkolů a opravování nahlášených defektů.
- Každý tým musí mít testera – tato role je zodpovědná za reportování defektů, kontrolu jejich opravy a následné uzavření problému.

7.2.8 Použitelný a nasaditelný produkt po každé iteraci

V rámci vývoje je žádoucí, aby docházelo k pravidelné integraci výsledků práce všech týmů. Přinejmenším na konci každé iterace musí být výsledný produktový přírůstek již plně integrován. Díky tomu lze tento plnohodnotný inkrement snadno prezentovat zákazníkovi a ostatním stakeholderům. Zároveň je dosaženo průběžného generování přidané hodnoty s každou iterací.

Symptomy

- Na konci každé iterace (v jejím posledním týdnu) existuje nová verze produktu (*release*).
- V nové verzi je zvýšeno hlavní či vedlejší číslo verze (major/minor označení).
- Nová verze obsahuje příslušné označení (*tag*) v repozitáři (např. „Release“).
- Přírůstek systému obsahuje programový kód (*commity*) od všech týmů.

7.2.9 Pravidelné týmové meetingy

Koncept samořídících týmů je jednou z nejdůležitějších charakteristik v kontextu agilních metodik. Aby týmy fungovaly skutečně samostatně, je nutné, aby všichni členové přesně věděli, kde se projekt aktuálně nachází. Také by měly být sjednoceny veškeré postupy, představy a očekávání. Proto je doporučeno organizovat pravidelné týmové meetingy.

Symptomy

- V průběhu projektu se konají pravidelné týmové meetingy (tzv. *standupy*).
- Tyto meetingy se konají ideálně na každodenní bázi, nejméně jsou však pořádány jednou týdně.
- Těchto koordinačních meetingů se účastní všichni členové týmu.

7.2.10 Průběžná údržba produktového backlogu

Všechny popisované frameworky doporučují věnovat jistý čas průběžné údržbě produktového backlogu (tzv. PBR – *Product Backlog Refinement*). Tato činnost by měla být prováděna v rámci plánování každé nadcházející iterace. V jejím průběhu dochází ke změnám vlastností položek produktového backlogu, případně jsou některé nové položky do tohoto backlogu přidávány.

Symptomy

- Existuje týmový meeting, v jehož názvu či popisu se nachází alespoň jeden z následujících řetězců:
  - „plánování“,
  - „planning“,
  - „PBR“,
  - „backlog refinement“,
  - „backlog grooming“.
- Tento meeting se koná v posledním týdnu předcházející iterace nebo v prvních dvou dnech iterace aktuální.
- V rámci údržby backlogu dochází k následujícím akcím:
  - úprava a přidávání položek či dílčích úkolů do backlogu,
  - změna priorit položek,
  - změna zodpovědné osoby,
  - změna časových odhadů.

7.2.11 Průběžná úprava dokumentace

Agilní projekty by se měly vyhnout tvorbě příliš detailních a komplexních aplikačních designů v úvodních fázích. Vhodnější je vypracovávat tyto dokumenty postupně i v konstrukčních fázích, díky čemuž mohou vývojové práce začínat mnohem dříve. Znatelně se tak zrychlí čas, který je nutný k dokončení dané funkcionality.

Symptomy

- Dokumenty a aplikační designy jsou průběžně doplňovány a upravovány.
- Změny dokumentů a aplikačních designů jsou prováděny i v konstrukčních fázích, nikoli pouze v těch úvodních.

7.2.12 Retrospektiva

Proces postupného zlepšování je v kontextu agilních metodik velmi důležitý. Všechny metodiky popisují nezbytnost průběžné kontroly a následného vylepšování prováděných činností a postupů (přístup *Inspect & Adapt*). Klíčovým artefaktem se tak stává retrospektiva, na jejímž základě dochází ke zhodnocení iterace a identifikují se činnosti a postupy, které mohou být v rámci příštích iterací zlepšeny.

Symptomy

- Existence takového artefaktu, který obsahuje řetězec „retrospe“ ve svém názvu, popisu nebo obsahu. Tento řetězec byl zvolen z důvodu ustáleného pojmenování této činnosti na základě iterativních a agilních metodik.
- Tento artefakt by měl být tvořen (či upravován) v posledním týdnu každé iterace, případně v prvních dvou dnech iterace nadcházející.

7.2.13 Velikost týmu

V rámci agilních týmů je doporučována co nejvyšší míra dekompozice projektu. Týmy by tak měly mít poměrně nízký počet členů (kteří by navíc měli pracovat co nejblíže u sebe – ideálně v jedné místnosti). To platí dvojnásob pro rozsáhlé projekty. Je tedy možné definovat maximální horní hranici pro velikost týmu.

SAFe jasně definuje, že počet členů týmu by se měl pohybovat mezi 5 a 11 lidmi. LeSS explicitně žádná jasná čísla neuvádí, avšak při rozhodování pro přechod na LeSS Huge jsou zmiňovány osmičlenné týmy, což není v rozporu ani s klasickým Scrumem. S DAD je to však v této oblasti komplikovanější. Tento framework totiž připouští i týmy čítající více než 30 členů. V některých případech jsou však týmy již od velikosti přibližně 10 členů považovány spíše za středně veliké.

Symptomy

- Počet členů žádného z týmů by neměl přesahovat 12 lidí.
- V tomto počtu jsou zahrnuti pouze členové zpracovatelského týmu – vývojáři, analytici, testeři atd. Nejsou zde zahrnuty osoby ani role, které v hierarchii stojí mimo úroveň týmu (např. manažeři nebo konzultanti).

Možné vylepšení

Navržené řešení může být v mnoha případech příliš přísné. Např. v případě, kdy by měl projekt 10 týmů, přičemž 9 z nich by bylo osmičlenných a jeden by byl větší – třináctičlenný. Proto by mohlo být vhodné upravit symptomy následujícím způsobem:

- 80 % týmů smí mít maximálně 12 členů.
- Žádný tým nesmí mít více než 15 členů. Maximální hranice byla takto nastavena proto, že ani nejvíce benevolentní framework DAD nedovoluje malým týmům čítat více než 15 lidí.

7.2.14 Vizualizace a vizuální modelování

Z grafů, obrázků, modelů a náčrtů lze mnohdy nejsnáze pochopit některé komplikované situace, postupy a vztahy mezi jednotlivými objekty. Diskuze v rámci týmu jsou často doplňovány znázorňováním myšlenek např. na tabule. Mohou být samozřejmě zachyceny i pomocí jiných nástrojů (manuálně či digitálně). Tyto vizuální výstupy by pak měly být zachyceny a příslušně uchovány ve sdíleném úložišti pro potřeby možného dalšího budoucího využití.

Symptomy

- V projektu lze takovéto artefakty vyhledat podle typu MIME (*Multipurpose Internet Mail Extensions*).
- Artefakty lze rozpoznat také podle přípony – obrázky, PDF apod.
- V názvu souboru nebo v jeho popisu lze vyhledat řetězce „diagram“ nebo „model“.
- Tyto soubory by měly vznikat převážně v počáteční fázi projektu.

8 ZJIŠTĚNÉ VÝSLEDKY

V této kapitole jsou nejprve porovnány všechny 3 popisované frameworky. Důraz je kladen zejména na popis situací, ve kterých je vhodné zvolit jeden z těchto frameworků. V druhé části kapitoly jsou podrobně rozebrány výsledky detekce praktik pomocí nástroje SPADe na sadě testovacích dat a jsou popsány odhalené problémy.

8.1 Srovnání frameworků

Všechny tři popisované frameworky toho mají mnoho společného. Všechny se věnují především specifikům rozsáhlých projektů a distribuovaným týmům. Všechny jsou také postaveny na myšlenkách a principech agilního vývoje software. Příliš se nevěnují nízkoúrovňovým procesním problémům a postupům. Zato poskytují sadu myšlenek a principů, kterými by se měl projektový management řídit a měl by dbát na jejich dodržování.

Základem všech popisovaných frameworků je Scrum. Dále jsou pak u některých frameworků využívány i různé poznatky z UP (*Unified Process*), XP (*Extreme Programming*), Lean atd. Nemá tedy příliš smysl porovnávat tyto metodiky s popisovanými frameworky, jelikož se jedná pouze o jejich dílčí části, ze kterých jsou navíc čerpány pouze vybrané koncepty. Poznatky z klasických metodik jsou tedy využívány pouze pro některé procesní části projektového řízení v rámci rozsáhlých projektů.

8.1.1 SAFe

Framework SAFe popisuje 3 své základní úrovně – tým, program a portfolio. To je velká změna oproti např. Scrumu, který takovéto rozložení společnosti vůbec neřeší. Zároveň to však naznačuje existenci jisté vertikální struktury, což není zcela v souladu s agilními principy. V předchozích verzích SAFe se to projevovalo zejména na týmové úrovni, kde jednotlivé týmy nemusely být skutečně multifunkční. Obsahovaly totiž zejména vývojáře a testery, přičemž analytici se centralizovali především na programové úrovni. Tím docházelo k omezení kreativity a přítomnosti know-how v rámci samotných týmů. Tento zásadní problém byl však v posledních verzích frameworku SAFe napraven.

SAFe však bývá často kritizován i za další rozpory v porovnání s agilním manifestem. Jedná se zejména o přílišnou striktnost a definici procesů. Ty potlačují samořídící schopnosti jednotlivých týmů a fungují jako prvek mnoha restrikcí. SAFe se totiž ve svých počátcích výrazně inspiroval metodikou UP (mnohem více než ostatní popisované frameworky). Tato inspirace je ve frameworku i po několika evolučních cyklech stále znatelná. Kvůli těmto nedostatkům čelil framework v minulosti mnoha kritikám od agilních

expertů [31] [32]. Všichni se zpravidla shodují, že SAFe není plně v souladu s agilními hodnotami.

Tyto odlišnosti od agilních zásad jsou však zároveň i jednou z velkých výhod SAFe. Plné osvojení komplexního agilního přístupu je totiž pro mnoho společností velmi obtížné a trvá to i několik let. Jelikož SAFe se v mnoha ohledech podobá strukturám stávajících organizací, a navíc je poměrně striktní v definici svých procesů, jedná se o ideální framework pro učinění prvního kroku k přechod od sekvenčního přístupu k agilnímu vývoji.

8.1.2 LeSS

Oproti SAFe je LeSS zcela agilní, minimalistický a flexibilní framework, který je plně v souladu s agilním manifestem a jeho stěžejními principy. Vychází totiž plně ze Scrumu a pouze jej doplňuje o sadu rozšíření, která jsou nezbytná pro řízení více týmů a rozsáhlých projektů. Díky tomu jej lze považovat za nejvíce agilní framework ze všech popisovaných. Výhodou také je, že týmy, které již úspěšně implementovaly Scrum, nemusí pocítit při přechodu na LeSS žádnou výraznou změnu. Na úrovni týmů se totiž téměř nic nemění. Tento framework je tak vhodný pro organizace, které si již Scrum (či jinou agilní metodiku) osvojily. Výhodou je také existence dvou typů frameworku pro různě rozsáhlé projekty.

Framework má však stejnou zásadní nevýhodu, která pramení z filozofie Scrumu. Není příliš preskriptivní a záměrně nechává spoustu otevřených otázek, které by si měla každá organizace či tým vyřešit a uzpůsobit vlastním potřebám. Mnoho principů tak zůstává pouze jednoduše popsáno bez jakéhokoli návodu, jak dané principy realizovat a dodržet. To může být zásadní problém pro organizace, které nemají zkušenosti s agilním vývojem. Tím je pouze potvrzena předchozí teze, že tento framework je vhodný pro společnosti, které již agilní přístup dokázaly implementovat.

8.1.3 DAD

DAD se taktéž snaží o doplnění některých částí, o kterých standardní Scrum mlčí. Na rozdíl od LeSS to však není pojato pouhým rozšířením Scrumových praktik, ale dochází zde k definici praktik nových. Obdobně jako SAFe se inspiruje i mnoha dalšími metodikami, přístupy a myšlenkami. To se projevuje např. v dělení projektů do fází (zahájení, konstrukce, předání), což vychází z metodiky UP. To není zcela v souladu s agilním přístupem z důvodu existence úvodní zahajovací fáze.

Tento framework je však koncipován velmi specificky. Jedná se o rozhodovací framework, který je zaměřen na dosažení cílů. DAD nepředepisuje žádné praktiky, které by měly být striktně dodržovány. Místo toho vždy nabízí sadu alternativ, mezi kterými si může každý projekt zvolit podle svých vlastních potřeb. U každé alternativy je pak popsána její

charakteristika, výhody, nevýhody a aspekty, kterým je třeba věnovat zvýšenou pozornost a opatrnost. Je tak pokryta široká škála nejrůznějších projektových požadavků a každý projekt si může framework přizpůsobit na míru. Není tedy nutné stanovovat jednotné a striktní postupy pro fungování všech týmů v rámci celé organizace.

Tato volnost však může být zároveň i nevýhodou. Jelikož je totiž definována pouze minimální sada vodítek, může být výběr vhodných postupů velice komplikovaný a obtížný. Je tedy zcela nezbytné, aby projektový management přesně znal budoucí potřeby projektu a aby si uvědomoval jeho kompletní kontext i v rámci organizace a její struktury. Není zde takový tlak na předchozí znalost agilního přístupu (jako tomu je u frameworku LeSS), avšak o to větší tlak je kladen na expertízu projektového vedení. Je třeba volit korektní postupy pro reálné potřeby projektu.

8.2 Detekce praktik pomocí SPADe

Detekovatelné praktiky z kapitoly č. 7.2 je potřeba ověřit na datech z reálných projektů. Díky této kontrole lze odhalit případné nedostatky při definici praktik. Také je možné zjistit další okolnosti, kvůli kterým může být výsledná detekce zkreslená či neúplná. Jedná se o jediný způsob kontroly správnosti sestavených dotazů, pomocí kterých jsou tyto praktiky odhalovány.

Kontrolu je třeba provést nad sadou testovacích dat. K tomuto účelu byla využita data získaná v průběhu předmětu KIV/ASWI (Pokročilé softwarové inženýrství) z roku 2017. V této sadě se nachází informace o celkem 12 projektech. Jedná se o malé školní projekty, které se skládají z jednoho týmu a zpravidla čítají 4–5 lidí.

Tato data tak nejsou zcela vhodná pro metodiky popsané v této práci. Vhodnější by byly údaje o větších projektech, na kterých se podílelo více týmů. Taková data však bohužel nebyla k dispozici. Dostat se k open-source projektovým datům je poměrně komplikované, zejména pak pokud se má jednat o rozsáhlé projekty s distribuovanými týmy.

Pro účely ověření funkčnosti dotazů byly vybrány 3 agilní praktiky. Konkrétně se jedná o kontrolu korektní a stabilní délky iterace (viz kap. 7.2.2), tvorbu retrospektivy (viz kap. 7.2.12) a pořádání pravidelných týmových meetingů (viz kap. 7.2.9). Tyto praktiky byly zvoleny zejména proto, že je jejich detekce z velké části možná i v rámci malých projektů z testovací sady.

8.2.1 Délka a stabilita iterace

V první fázi byla spočítána délka jednotlivých iterací u každého projektu. Toho bylo dosaženo odečtením startovního data iterace od koncového data iterace. Pokud se výsledný

počet dní pohyboval v rozmezí mezi 7 a 21 dny, iterace měla vhodnou délku. Projekt splňuje tuto praktiku, pokud ji splňují i všechny dílčí iterace.

Při kontrole stability délky iterací byl nejprve vypočten základ pro určení standardní doby trvání jednotlivých iterací. Ten se rovná nejčastěji zjištěné délce iterací v rámci projektu. Pokud toto číslo nelze jednoznačně určit, je použit průměr všech délek. Tento zjištěný práh je použit při určení samotné stability doby trvání iterací. Pokud je délka iterací maximálně o 3 dny kratší či delší než vypočtený práh, je její délka považována za stabilní. Část výsledků je zachycena v tab. 8.1.

V některých případech mohou existovat kratší počáteční nebo finální fáze projektu. To by bylo možné tolerovat, jelikož časové nároky na tyto iterace nejsou příliš veliké – zpravidla se může jednat pouze o formální zahájení či ukončení projektu. V průběhu projektu také může dojít ke změně ve standardní délce iterací. To může nastat v případě, že původně nastavená délka nevyhovuje požadavkům týmu a je potřeba ji přehodnotit. Obě tyto situace jsou sice validní, nicméně v rámci sestavených dotazů k nim v současné době není přihlíženo.

Z testovacích dat vyplývá, že všechny týmy převážně dodržovaly korektní délku iterací (viz soubor *vysledky_detekce.xlsx*, sešit *Délka a stabilita iterací* na přiloženém CD). Pouze v ojedinělých případech tato délka vybočovala ze stanoveného intervalu, avšak zpravidla se tak stalo v první či poslední iteraci. To je akceptovatelné. Převážná většina iterací také měla stabilní délku. Lze tedy tvrdit, že všechny týmy tuto praktiku z velké části dodržovaly.

Projekt	**Pořadí iterací**	**Délka iterace [dny]**	**Detekce korektní délky**	**Práh stability [dny]**	**Detekce stability**
Projekt 1	1	10	1	9	1
	2	21	1		0
	3	14	1		0
	4	9	1		1
	5	9	1		1
Projekt 2	1	13	1	14	1
	2	14	1		1
	3	14	1		1
	4	14	1		1
	5	14	1		1
	6	27	0		0
Projekt 3	1	10	1	7	1
	2	7	1		1
	3	7	1		1
	4	7	1		1
	5	7	1		1

Tab. 8.1: Část výsledků detekce korektní délky a stability iterací

8.2.2 Existence retrospektivy

V rámci prvního kroku je nejprve nutné zjistit počet iterací v projektu. Tato informace je potřebná, jelikož je známé, že každá iterace by měla být retrospektivou zakončena. Retrospektiv by tedy mělo být alespoň tolik, kolik bylo v projektu iterací.

Zásadní překážkou je nejednotnost způsobu tvorby retrospektivy. Každý projekt může tuto činnost provádět odlišně. Proto byly specifikovány 3 hlavní způsoby, pomocí kterých lze existenci retrospektiv odhalit. Prvním způsobem je kontrola existence ticketu s řetězcem „retrospe“ v názvu. Další 2 způsoby se vztahují k hledání stejného řetězce v názvech artefaktů a také v jejich obsahu. Následně dojde k sečtení všech nalezených výskytů pro každý ze způsobů. Aby bylo dodržování této praktiky potvrzeno, musí být součet alespoň jednoho z definovaných způsobů větší nebo roven počtu iterací (viz tab. 8.2).

V rámci detekce této praktiky je pracováno s předpokladem, že se způsob tvorby retrospektivy v průběhu projektu nemění. V opačném případě by bylo obtížné odhalit vzájemné vztahy mezi detekovanými výskyty a problém by nastal také s ignorováním duplicit.

Jedním z hlavních úskalí navrženého řešení může být špatná detekce podle zvoleného řetězce. Pojmenování hledané činnosti či artefaktu se může u různých projektů lišit (např. v závislosti na používaném jazyce), případně může být také daný řetězec použit ve zcela jiném kontextu. Také není zohledněno, zda se retrospektivy prováděly v korektní dobu – na konci každé iterace.

Další odhalený nedostatek souvisí s používáním nástroje Redmine v rámci zkoumaných projektů. Ten pro ukládání souborů zpravidla využívá DMS (*Document Management System*) plugin. Datová pumpa nástroje SPADe však s pluginy v tuto chvíli nedokáže pracovat, a proto data z nich nemohla být přenesena do datového skladu. Dokumenty uložené v DMS tak nemohou být v průběhu detekce nalezeny ani zohledněny.

Po provedení detekce nad testovacími daty je zřejmé, že pouze polovina týmů dodržovala pravidelnou tvorbu retrospektivy (viz soubor *vysledky_detekce.xlsx*, sešit *Retrospektiva* na přiloženém CD). U dvou týmů však nebyly retrospektivy dohledány kvůli problému s DMS. U dalších 2 týmů nedošlo detekování praktiky, jelikož všechny retrospektivy byly ukládány do jednoho artefaktu (ve kterém bylo hledané slovo zmíněno pouze v nadpisu).

8.2.3 Pravidelné týmové meetingy

Pravidelné týmové meetingy jsou velmi důležitou součástí agilních metodik a samořídících týmů. V ideálním případě by měly být prováděny na každodenní bázi. K tomu však v kontextu testovaných projektů nemohlo docházet, a proto byl pro účely detekce zvolen týdenní interval.

Projekt	Počet iterací	Tickety	Názvy artefaktů	Obsah artefaktů	Detekce retrospektivy
Projekt 1	5	1	5	10	1
Projekt 2	6	4	1	7	1
Projekt 3	4	1	1	2	0
Projekt 4	6	0	0	5	0
Projekt 5	6	0	0	2	0
Projekt 6	5	4	4	0	0
Projekt 7	7	1	3	8	1
Projekt 8	5	0	1	2	0
Projekt 9	6	9	8	9	1
Projekt 10	6	0	0	2	0
Projekt 11	5	1	5	2	1
Projekt 12	9	6	7	9	1

Tab. 8.2: Výsledky detekce existence retrospektivy

Nejprve tedy došlo k rozdělení projektů po jednotlivých týdnech. Následně začalo prohledávání všech artefaktů a ticketů, které byly v daném týdnu vytvořeny, editovány či okomentovány. Bylo potřeba vybrat vhodný řetězec, který bude v projektových datech vyhledáván. Na jeho základě by mělo být možné určit, zda se týmový meeting konal. Obecně jsou tyto pravidelné meetingy v agilních metodikách nazývány slovem *standup*. Jedná se o unikátní pojem, který přesně vystihuje vyhledávanou činnost. Část výsledků detekce se nachází v tab. 8.3.

Pro účel vyhledávání byl tedy nejprve zvolen řetězec „stand“. V takovém případě však bylo generováno mnoho falešně pozitivních nálezů. To bylo zapříčiněno zejména shodným základem se slovem „standard“. Proto následně došlo k větší konkretizaci výrazu. Byla tedy vyhledávána skupina řetězců „standup“, „stand-up“ a „stand up“. Pokud je alespoň jeden takovýto výskyt detekován, lze tvrdit, že tato praktika byla v daný týden dodržena. Celý projekt praktiku dodržoval, pokud byla úspěšně detekována pro každý dílčí týden.

Problémy při detekci mohou i zde souviset s vyhledáváním konkrétních řetězců – některý projekt může tyto meetingy nazývat zcela odlišně. Přidávání dalších řetězců však může velice negativně ovlivnit výsledky detekce, jelikož mohou být využívány i v jiných kontextech. Např. slova „meeting“ nebo „schůzka“ mohou být použita i pro setkání se zadavatelem nebo s garantem.

Jelikož byla testovací data získána ze školních projektů, tak meetingy mohly v jisté formě probíhat, přičemž o nich pouze nebyl proveden patřičný záznam. Z detekce by také mohl být vyřazen první a poslední týden. Jelikož tyto týdny mohly být kratší, meetingy ještě nemusely být řádně naplánovány, případně již nemusely být potřeba.

Z testovacích dat vyplývá, že pouze u 2 týmů bylo dodržování této praktiky detekováno (viz soubor *vysledky_detekce.xlsx*, sešit *Týdenní meetingy* na přiloženém CD). Ostatní týmy pravidelné meetingy buď nepořádaly, nebo o nich nevytvářely zachytitelné záznamy. To potvrdila i následná manuální kontrola.

Projekt	Kalendářní týden	Počet výskytů vyhledávaných řetězců	Detekce meetingu
Projekt 1	10	0	0
	11	4	1
	12	3	1
	13	3	1
	14	3	1
	15	0	0
	16	2	1
	17	0	0
	18	0	0
	19	0	0
	20	0	0

Tab. 8.3: Část výsledků detekce pořádání pravidelných týmových meetingů

8.2.4 Zhodnocení výsledků detekce

Pomocí nástroje SPADe byly detekovány 3 vybrané agilní praktiky – korektní a stabilní délka iterací, existence retrospektiv a pořádání pravidelných týmových meetingů. Tyto praktiky byly zkoumány v sadě testovacích projektů. Byl popsán zvolený postup při detekci, její výsledky a zjištěné problémy.

Základním problémem při detekci je rozmanitost a nekonzistentnost dat ve zdrojových nástrojích. Každý projekt může stejnou činnost provádět mnoha různými způsoby, případně může také docházet k nesprávnému používání nástrojů. Nejvhodnějším způsobem, jak dosáhnout optimálních výsledků, je přizpůsobit dotazy každému projektu odděleně – ideálně pouze pomocí změny parametrů dotazu (např. vyhledávané slovo). Je tedy nutná detailní znalost postupů a praktik každého projektu ještě předtím, než bude snaha o detekci vůbec zahájena.

Existence nějakého artefaktu nebo ticketu také nezaručuje, že daná činnost skutečně proběhla a byla dokončena. Zjištěné výsledky je tak nutné chápat především jako indikaci, kterou je následně nezbytné porovnat s reálnými znalostmi o projektu.

Další překážku mohou představovat nejrůznější pluginy jednotlivých ALM nástrojů. Na ty není datová pumpa uzpůsobena, takže některá důležitá data mohou být opominuta.

9 ZÁVĚR

V práci bylo představeno několik moderních metodik a technik pro vývoj software. Z nich byla vybrána skupina tří metodik – Disciplined Agile Delivery (DAD), Large-Scale Scrum (LeSS) a Scaled Agile Framework (SAFe). Všechny tyto frameworky jsou uzpůsobeny potřebám rozsáhlých projektů a vycházejí zejména z principů agilního vývoje.

Následně došlo k podrobnému a konsolidovanému popisu zvolených metodik. Pro každou z nich byly zpracovány především dostupné knižní publikace, oficiální online znalostní báze a případové studie. U jednotlivých frameworků byly zdůrazněny jejich charakteristiky, základní principy, role, artefakty, činnosti a reálné dopady na několik skutečných projektů. Frameworky byly také porovnány vůči sobě navzájem a v rámci jejich popisu docházelo i k akcentaci podobností s metodikami Scrum, UP apod.

Na základě zjištěných poznatků byla definována sada praktik, které všechny popisované metodiky v různé míře dodržují. Tyto praktiky byly stručné popsány a u každé z nich došlo ke specifikování jejích konkrétních symptomů. Byl také popsán nástroj SPADe, jeho možnosti a využívaný doménový model. Definované symptomy tak mohly být z velké části uzpůsobeny možnostem tohoto nástroje.

V závěrečné části práce došlo ke srovnání popisovaných frameworků. Každý z nich se hodí pro jiné typy projektů a organizací, a proto byl kladen speciální důraz zejména na popis situací, ve kterých je vhodné zvolit jeden z nich. Také byly vyhodnoceny výsledky detekce vybraných praktik na základě dat z dostupných příkladových projektů. Díky tomu mohly být odhaleny některé problémy související se samotnou detekcí praktik, nekonzistentností zkoumaných dat, nedokonalostmi postupů v rámci jednotlivých projektů či problémy s nástrojem SPADe.

Tato práce může sloužit především pro získání povědomí o popisovaných frameworcích, moderních agilních technikách a o problémech, které souvisí se škálováním projektů v oblasti IT. Také může být využita při budoucím rozvoji experimentálního nástroje SPADe.

LITERATURA

[1] LARMAN, Craig. Agile and iterative development: a manager's guide. Boston: Addison-Wesley, 2004. 342 s. The agile software development series. ISBN 0-13-111155-8.

[2] Manifesto for Agile Software Development [online]. 2001 [cit. 2019-01-13]. Dostupné z: https://agilemanifesto.org/

[3] KIM, Gene et al. The DevOps handbook: how to create world-class agility, reliability, & security in technology organizations. First edition. Portland: IT Revolution, [2016], 2016. 437 stran. ISBN 978-1-942788-00-3.

[4] DevOps: Breaking the Development-Operations barrier. Atlassian.com [online]. 2019 [cit. 2019-02-09]. Dostupné z: https://www.atlassian.com/devops

[5] Introduction to Disciplined Agile Delivery (DAD) [online]. 2019 [cit. 2019-02-09]. Dostupné z: http://disciplinedagiledelivery.com/introduction-to-dad/

[6] The DSDM Agile Project Framework (2014 Onwards) [online]. 2019 [cit. 2019-02-09]. Dostupné z: https://www.agilebusiness.org/content/introduction-0

[7] Benefits of The GROWS Method [online]. 2019 [cit. 2019-02-09]. Dostupné z: https://growsmethod.com/

[8] Introduction to LeSS [online]. 2019 [cit. 2019-02-09]. Dostupné z: https://less.works/less/framework/introduction.html

[9] An Introduction to the Nexus Framework [online]. 2019 [cit. 2019-02-09]. Dostupné z: https://www.scrum.org/resources/introduction-nexus-framework

[10] OIKOSOFY Enterprise Agile Framework [online]. 2019 [cit. 2019-02-09]. Dostupné z: http://oikosofy.com/oikosofy-enterprise-agile-framework/#intro

[11] SAFe: framework for scaling Agile [online]. 2019 [cit. 2019-02-09]. Dostupné z: https://www.scaledagile.com/enterprise-solutions/what-is-safe/

[12] JACOBSON, Ivar et al. The essence of software engineering: applying the SEMAT kernel. Upper Saddle River: Addison-Wesley Professional, 2013. xliii, 300 stran. ISBN 978-0-321-88595-1.

[13] What is SEMAT? [online]. 2019 [cit. 2019-02-09]. Dostupné z: http://semat.org/what-is-it-and-why-should-you-care-

[14] LEFFINGWELL, Dean et al. SAFe reference guide: scaled agile framework for lean software and systems engineering. Boston: Addison-Wesley, [2018], ©2018. xv, 793 stran. ISBN 978-0-13-489286-3.

[15] SAFe for Lean Enterprises [online]. 2019 [cit. 2019-02-16]. Dostupné z: https://www.scaledagileframework.com/

[16] SAFe Case Studies [online]. 2019 [cit. 2019-02-16]. Dostupné z: https://www.scaledagileframework.com/case-studies/

[17] SAFe Case Study: Capital One [online]. 2019 [cit. 2019-02-18]. https://www.scaledagileframework.com/capital-one-case-study/

[18] SAFe Case Study: Intel [online]. 2019 [cit. 2019-02-18]. Dostupné z: https://www.scaledagileframework.com/case-study-intel/

[19] AMBLER, Scott W. a LINES, Mark. Disciplined agile delivery: a practitioner's guide to agile software delivery in the enterprise. Upper Saddle River: IBM Press, ©2012. xxvii, 513 s. ISBN 978-0-13-281013-5.

[20] Disciplined Agile - The foundation for business agility [online]. 2019 [cit. 2019-03-11]. https://disciplinedagiledelivery.com/

[21] Accelerate Your Business with Disciplined Agile [online]. 2019 [cit. 2019-03-11]. https://disciplinedagileconsortium.org/Disciplined-Agile-Case-Study

[22] Benefits of Agile Transformation at Barclays [online]. 2019 [cit. 2019-03-16]. Dostupné z: https://disciplinedagileconsortium.org/resources/Pictures/Case%20studies/Benefits%20of%20Agile%20Transformation%20at%20Barclays.pdf

[23] Transforming from traditional to disciplined agile delivery - Panera Bread [online]. 2019 [cit. 2019-03-16]. Dostupné z: https://disciplinedagileconsortium.org/resources/Pictures/Case%20studies/Panera%20Case%20Study.pdf

[24] LARMAN, Craig a VODDE, Bas. Scaling lean & agile development: thinking and organizational tools for large-scale Scrum. Upper Saddle River: Addison-Wesley, ©2009. xiv, 348 s. ISBN 978-0-321-48096-5.

[25] More with LeSS [online]. 2019 [cit. 2019-03-20]. Dostupné z: https://less.works/

[26] LeSS Case Studies [online]. 2019 [cit. 2019-03-20]. Dostupné z: https://less.works/case-studies/index.html

[27] BMW Group - LeSS adoption at a bavarian car manufacturer [online]. 2019 [cit. 2019-03-27]. Dostupné z: https://less.works/case-studies/bmw-group.html#BMWGroup

[28] Nokia Networks - Developing a High Capacity Network Gateway with LeSS [online]. 2019 [cit. 2019-03-27]. Dostupné z: https://less.works/case-studies/nokia-networks-high-capacity-network-gateway.html

[29] PATIL, Neha. Large Scale Scrum (LeSS). In: Alchetron [online]. 2019, 2018-04-02 [cit. 2019-03-27]. Dostupné z: https://alchetron.com/Large-Scale-Scrum-(LeSS)

[30] PÍCHA, Petr. Detecting software development process patterns in project data. Plzeň, 2018. Koncept disertační práce. Západočeská univerzita v Plzni.

[31] SCHWABER, Ken. UnSAFe at any speed. In: Ken Schwaber's Blog [online]. 2019, 2013-08-06 [cit. 2019-04-19]. Dostupné z: https://kenschwaber.wordpress.com/2013/08/06/unsafe-at-any-speed/

[32] JEFFRIES, Ron. SAFe - Good But Not Good Enough. In: RonJeffries.com [online]. 2014, 2014-02-27 [cit. 2019-04-19]. Dostupné z: https://ronjeffries.com/xprog/articles/safe-good-but-not-good-enough/

SEZNAM OBRÁZKŮ

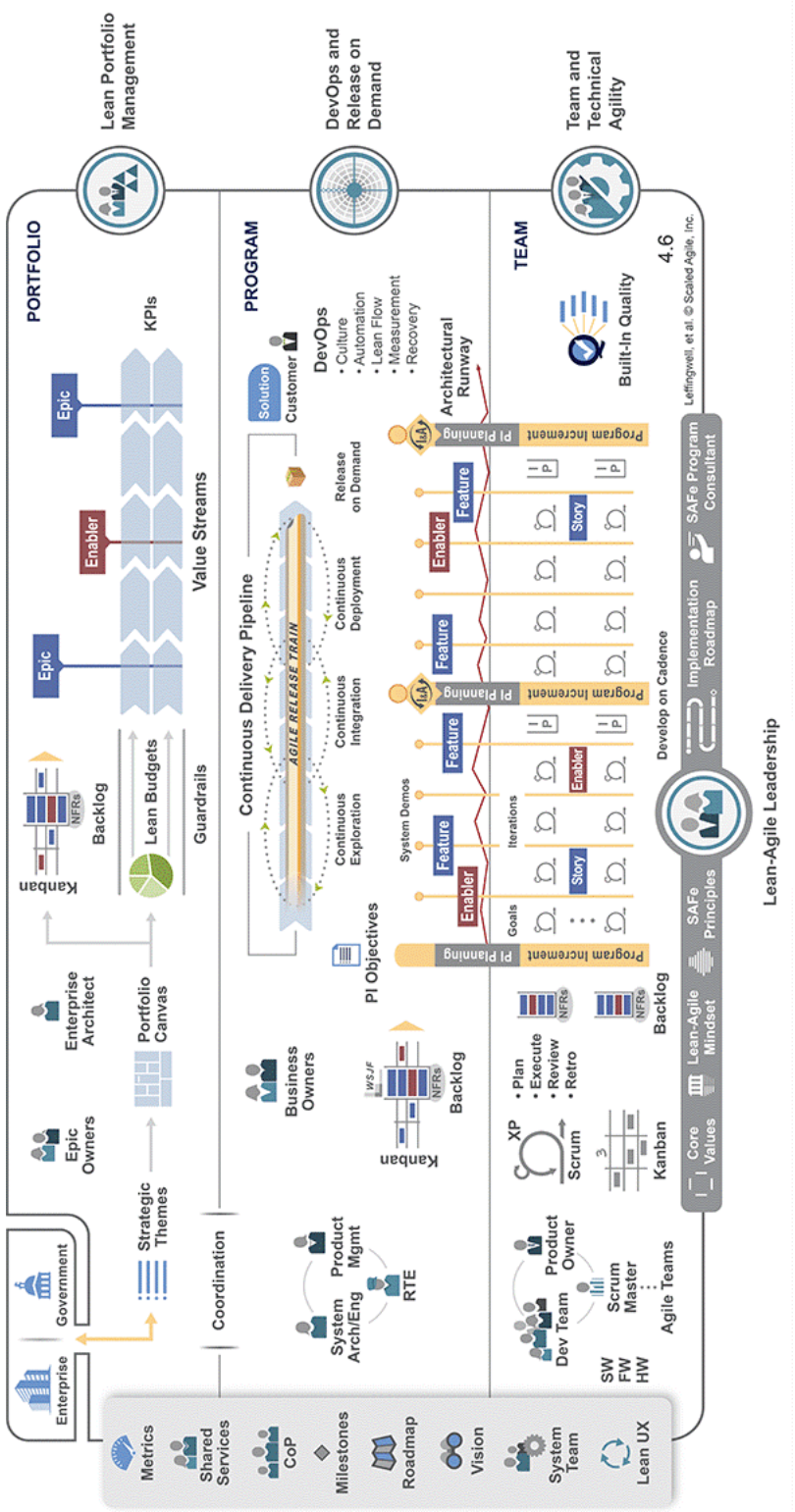
SEZNAM TABULEK

SEZNAM ZKRATEK

ALM	Application Lifecycle Management
ART	Agile Release Train
DAD	Disciplined Agile Delivery
DMS	Document Management System
LeSS	Large-Scale Scrum
RUP	Rational Unified Process
SAFe	Scaled Agile Framework
SPADe	Software Process Anti-pattern Detector
UP	Unified Process
XP	Extreme Programming

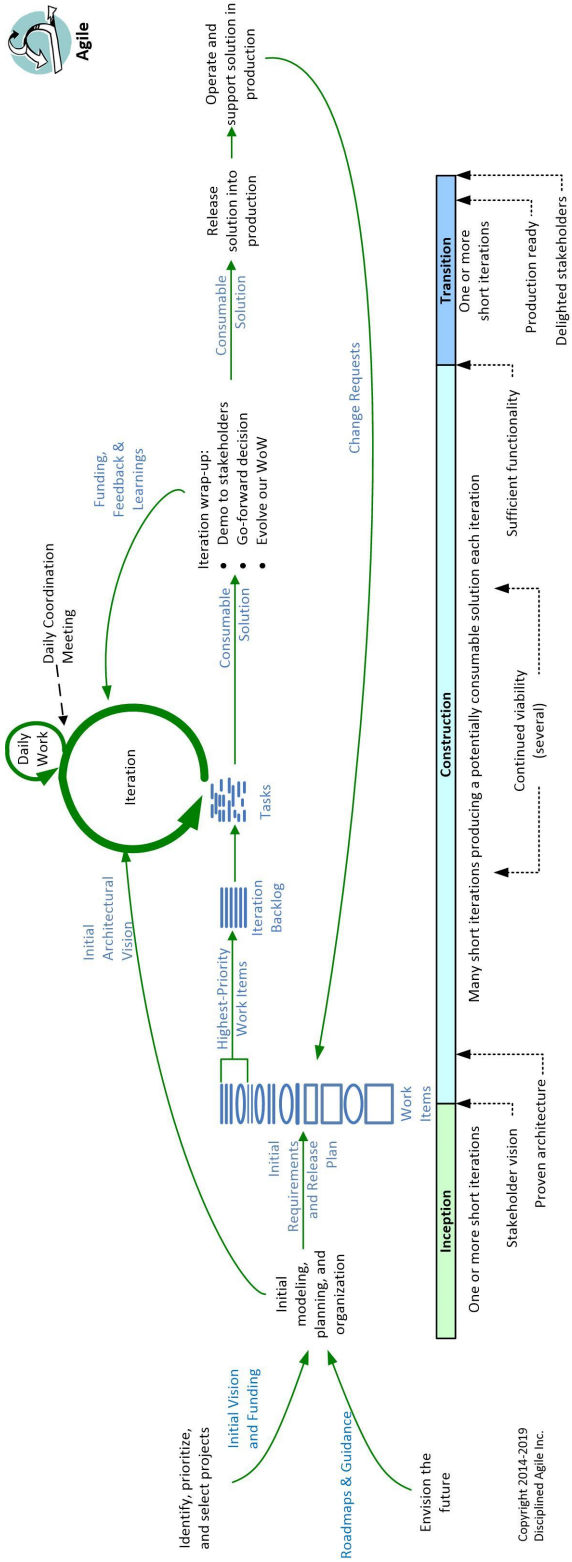
SEZNAM PŘÍLOH

A KONFIGURACE FRAMEWORKU SAFE



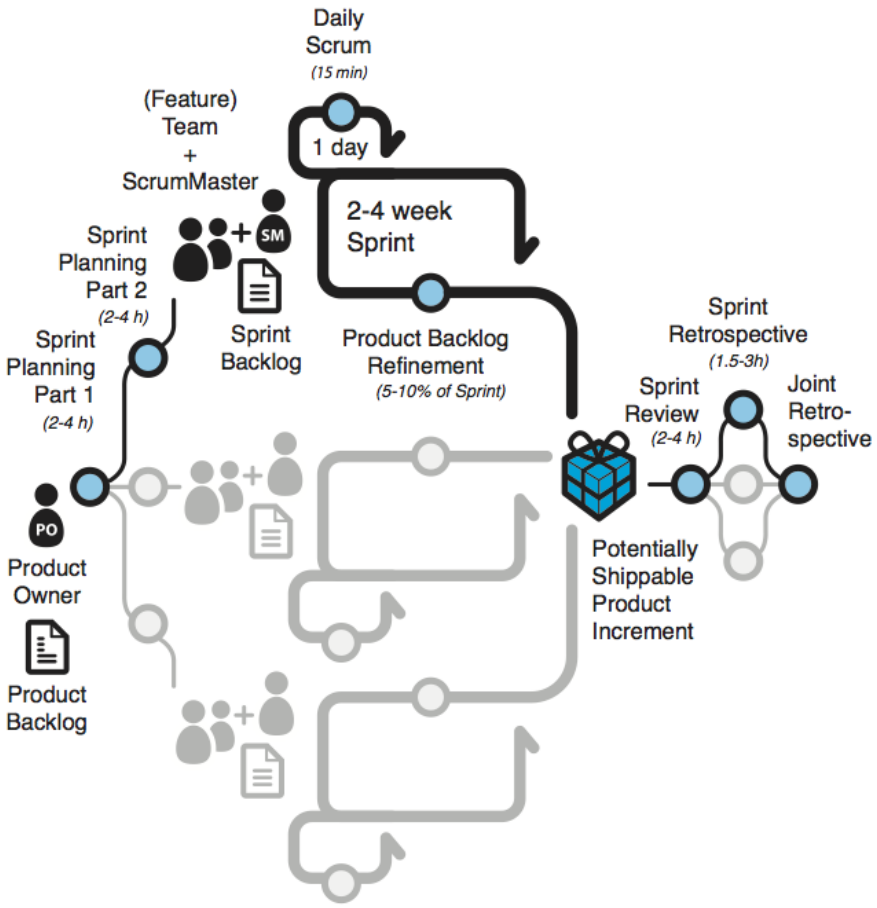
Obr. A.1: Konfigurace frameworku SAFe [15]

B AGILNÍ ŽIVOTNÍ CYKLUS DAD



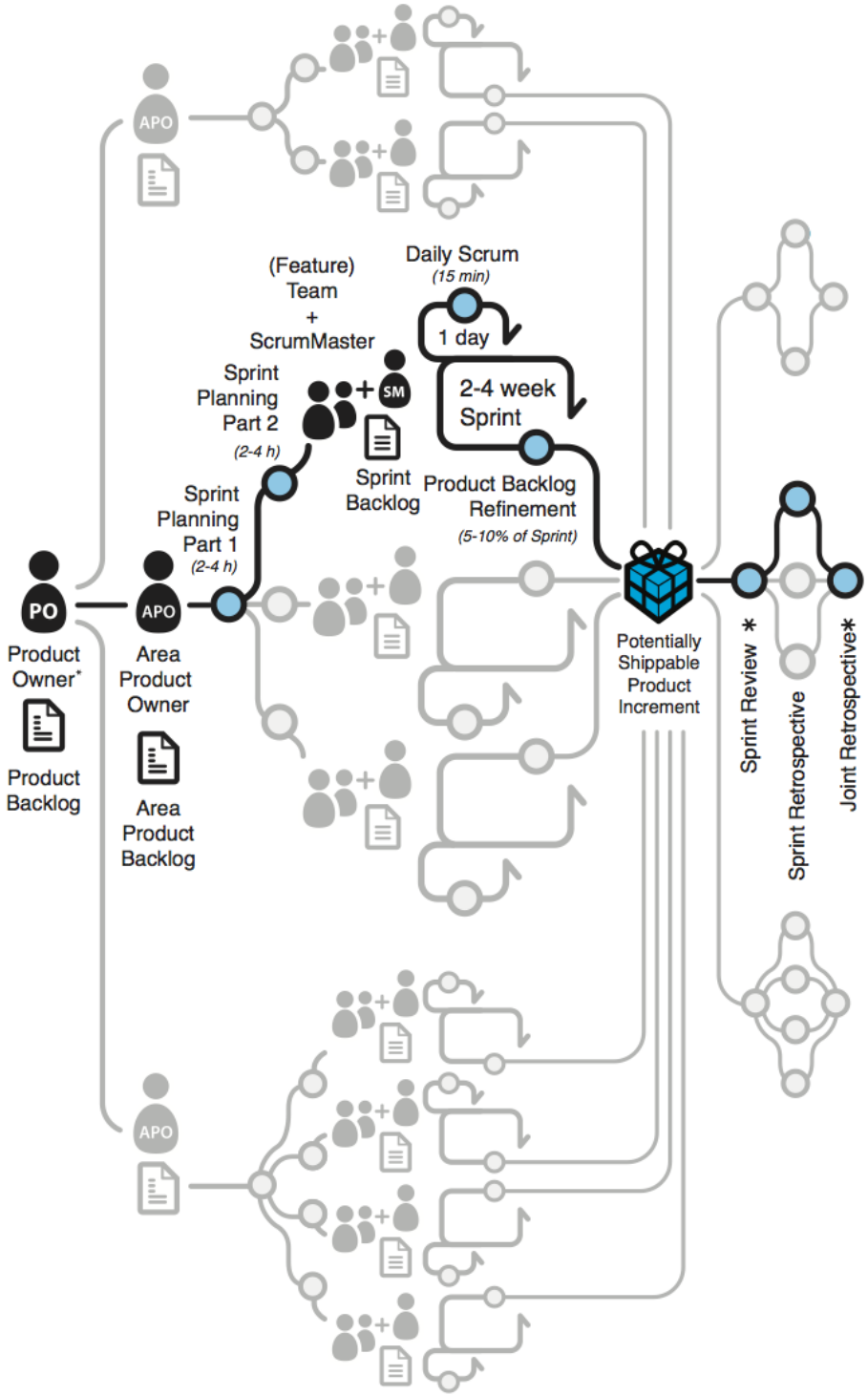
Obr. B.1: Agilní životní cyklus DAD [5]

C STRUKTURA PROJEKTU V LESS



Obr. C.1: Struktura projektu v LeSS [29]

D STRUKTURA PROJEKTU V LESS HUGE



Obr. D.1: Struktura projektu v LeSS Huge [29]