Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky

Katedra informačních technologií

Essence

Semestrální práce ke kurzu 4IT421 Zlepšování procesů budování IS	
Semestr	zimní 2014/2015
Autoři	Petr Kadlec, xkadp12 Lukáš Cibulka, xcibl00 Martin Kikerle, xkikm00 Róbert Schmidt, schr02 Kamil Prudí, xpruk00 Martin Šťastný, stam03
Téma	Essence
Datum odevzdání	19. 12. 2014

Obsah

Abstrakt

Essence je mladý standard, který byl publikován v listopadu 2014. Essence dle autorů není další metodikou, ale je to metodika na metodiku, teda metodika, která určuje jak pracovat s ostatními metodikami. Tenhle přístup není ve vývoji softwaru ničím novým, ale doposud se využíval jenom v praxi a chyběl mu teoretický resp. standartní základ, nad kterým by všichni vývojáři nebo jiní členové týmu mohli pracovat. Tahle práce si dává za cíl popsat standard Essence, co nové přináší, z čeho vznikl a jak se s ním pracuje.

Klíčové slova

Essence, SEMAT, kernel, jádro, alfa

1 Úvod

Essence je mladý standard, který byl publikován v listopadu 2014. (Object Management Group, 2014). Je to žhavé téma na poli metodik vývoje softwaru. Jelikož neexistuje k danému tématu mnoho literatury (jenom publikace autorů Essence a ta je dostupná jenom v anglickém jazyce) je vhodné se touto problematikou zabývat a hlouběji ji prozkoumat.

Essence dle autorů není další metodikou, ale je to vlastně metodika na metodiku, teda metodika, která určuje jak pracovat s ostatními metodikami. Tento přístup není ve vývoji softwaru ničím novým, ale doposud se využíval jenom v praxi a chyběl mu teoretický resp. standartní základ, nad kterým by všichni vývojáři nebo jiní členové týmu mohli pracovat. To je také jeden z důvodů, proč se danou problematikou zabývat, protože to rozšiřuje znalosti a přináší nové nápady do procesů vývoje informačních systémů a vedení projektů tohoto typu.

1.1 Cíl práce

Cílem práce je:

- Popsat a stručně charakterizovat Essence, z čeho vznikl a k čemu slouží
- Jak se s Essence a jeho jádrem pracuje
- Jak je možné Essence využít v práci v týmu nebo při vývoji softwaru
- Určit, co nové tento standard přináší.

1.2 Postup řešení

Práce vychází jenom z knihy dr. Jacobsona[1] a jeho přednášky[2], která je zestručněním jeho knihy. Důvod je ten, že touhle problematikou (Essence) se zabývali jenom členové OMG, tj. zejména dr. Jacobson a jeho tým.

K tomu abychom splnili stanovené cíle, jsme se rozhodli rozdělit jednotlivé kapitoly knihy mezi členy týmu. Každý člen tak bude zpracovávat samostatně určité téma (písemně) a studovat problematiku témat ostatních členů. Tak by měla být zajištěná zastupitelnost a případně vyřešeny nesrovnalosti.

[1] (Jacobson, a další, 2013)
[2] (Jacobson, a další, 2014)

2 SEMAT a jeho prospěšnost[3]

Dlouhodobě softwarové inženýrství trpí neuspokojivou statistikou úspěšnosti projektů. V prosinci roku 2009 byla zahájena iniciativa přebudování kvality softwarového inženýrství pod názvem Software Engineering Method and Theory, zkráceně SEMAT. Za tímto účelem vznikl Essence a podíleli se na tom zejména Ivar Jacobson, Bertrand Meyer a Richard Soley.

Essence se zabývá nedostatkem základních informací nezbytných pro úspěšné dokončení projektu, množstvím unikátních, agilních metod a podporuje škálovatelnost. Tradiční metody jsou definovány myšlenkou jak vytvořit instanci, případně činnost, jejímž cílem je dosažení výsledku podle přesné definice. Takové přístupy jsou obvykle vnímány v softwarovém inženýrství za kritické, nepřehledné a nepružné, což vede k tomu, že softwaroví inženýři se přiklánějí k metodě pokus - omyl a intenzivní spolupráci.

Mezi nejvýznamnější problémy při softwarovém inženýrství můžeme považovat:

- Multidimenzionalitu - Problémy mohou přijít z různých dimenzí, ať už od požadavků, architektury, týmové práce, schopností jednotlivců. Často se stává, že problémy jsou rozloženy do více dimenzí současně.
- Schopnosti lidí - komunikace na projektu, se zákazníkem ale i s jednotlivými dalšími týmy jsou klíčové úkoly. Často selhává komunikace lidí s různými znalostmi, proto se doporučuje mít kompetentní osobu pro každou dimenzi.
- Týmová disciplína - Obvykle vytvořit skvělý netriviální software vyžaduje značnou personální sílu a efektivní spolupráce je naprosto klíčová.

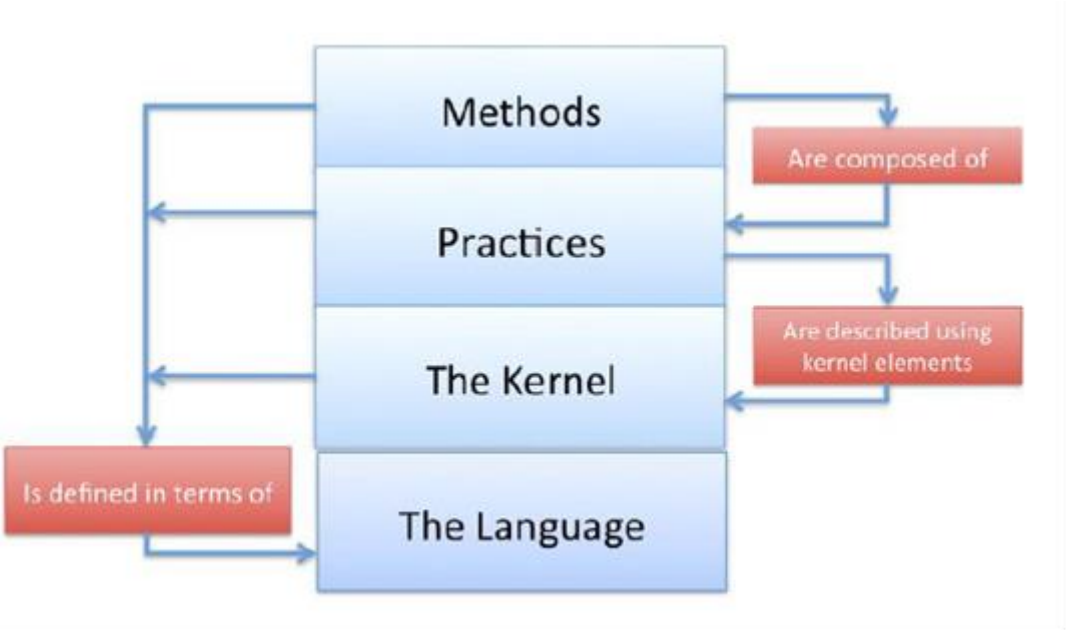
Essence identifikuje několik směrů softwarového úsilí a základem je, že tým musí dbát na posouzení pokroku. To zahrnuje požadavky, příležitosti, zúčastněné strany, práci, způsob práce a samozřejmě dodávku softwaru. Jádro Essence obsahuje příručku jak s každou z těchto věcí efektivně pracovat. Vyzdvihuje viditelnost kompetencí, které tým potřebuje pro správný přístup k softwarovému řešení. Identifikuje aspekty, kterým je potřebné se věnovat, aby byl vidět pokrok a dobrý stav projektu a zjednodušeně přináší návod jak vyřešit typické problémy softwarového vývoje. Koncepty v jádru nejsou zásadně nové, ale jsou intuitivní pro softwarové profesionály. Slouží jako společný základ pro softwarové projekty všech velikostí a jeho výhodou jsou nízké vstupní náklady s minimálním počtem překážek pro přijetí.

Je třeba si uvědomit, že Jádro je rozšiřitelné a stále ve vývoji. To znamená, že přináší neustále nové postupy, reaguje na nové technologie a sdílí nové zkušenosti ze softwarového vývoje.

2.1 Architektura Essence

Základem Essence je jednoduchá vrstevnatá architektura, kde jsou metody složené z oddělitelných postupů popsaných pomocí Jádra. Výrazně podporuje škálovatelnost, což přináší vývojářům možnost přizpůsobit si řešení pro unikátní situace a vytvořit efektivní metody.

[3] Zdroj této kapitoly je (Jacobson, a další, 2013)



Obrázek 1 Architektura metodiky

- Metody - Jejich stavebním kamenem jsou postupy. Podporují "day-to-day" činnosti a to je odlišuje od klasické definice. V popisu se nenachází jen, co se od metody očekává, ale co je skutečně z metody hotové i když není ještě kompletně dokončena.
- Praxe - Opakovaná snaha s konkrétním cílem. Praxe poskytuje systematický a ověřený postup jak řešit jednotlivé aspekty úlohy a může být součástí několika metod.
- Jádro - Obsahuje základní prvky a postupy softwarového inženýrství.
- Jazyk Essence - Doménový specifický jazyk, pomocí kterého definujeme metody, postupy a Jádro pro konkrétní projekt.

2.2 Základní prvky jádra (kernelu)

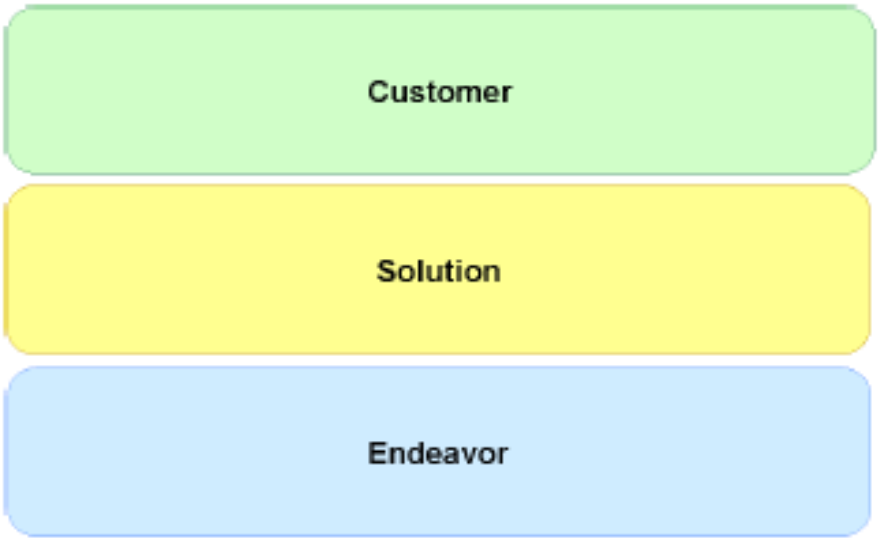
Jádro zachycuje sadu konceptů a definic, které jsou podstatné pro vývoj softwaru. Pomáhá komunikovat efektivně s týmem, překonávat problémy a vyvíjet efektivně. Tyto koncepty nejsou úplně nové, ale jsou intuitivní pro softwarové profesionály. Přináší výstižné, přehledné koncepty spolu s tím, jak je správně využívat. Jádro je však dynamické, často se mění a slovník, který používá v softwarovém inženýrství je také často upravován.

Toto je jen lehký úvod do přehledu elementů Jádra. Najít správné elementy může být náročné, neboť musí být jednoznačné, akceptovatelné, důležité, relevantní a musí se standardizovat.

Základní prvky jádra jsou:

- Alfy
- Místa aktivit
- Kompetence

Organizace Jádra se dělí na tři celistvé oblasti zájmu. Každá oblast se zabývá specifickou dimenzí softwarového inženýrství.



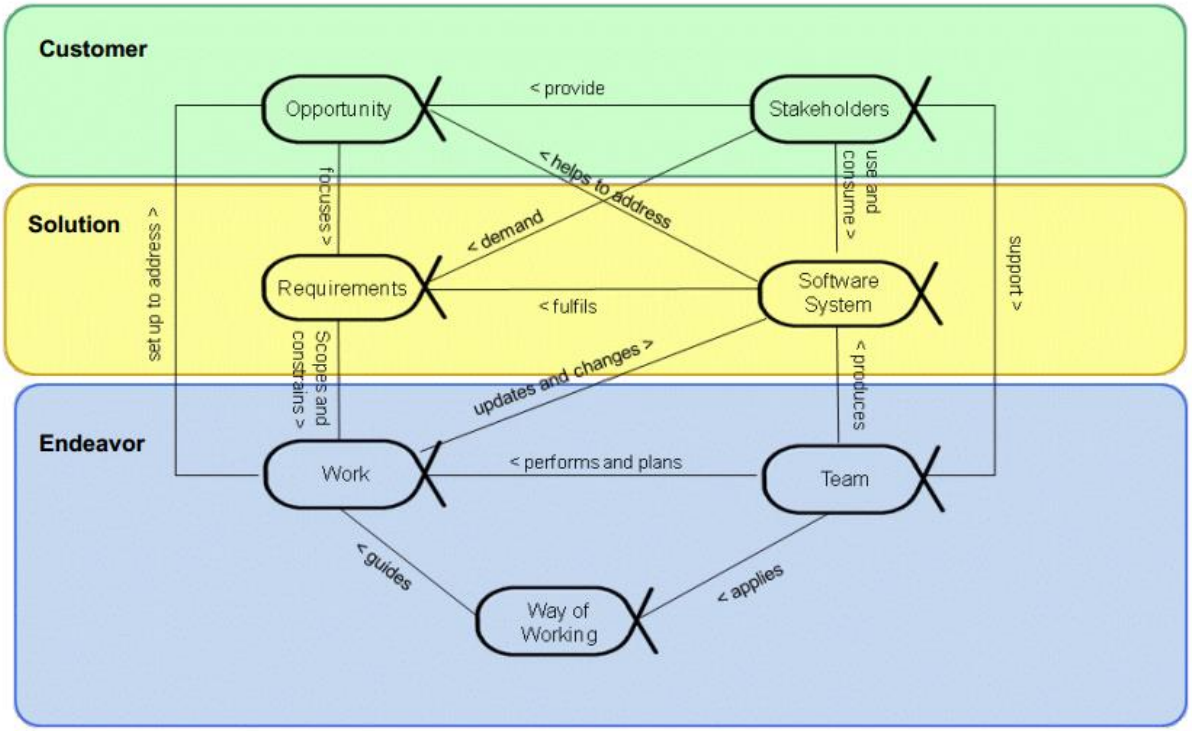
Obrázek 2 Organizace Jádra (kernelu)

- Zákazník - Vývoj softwaru vždy zahrnuje minimálně jednoho zákazníka, pro kterého je produkován. Pohled zákazníka musí být zahrnut v každodenní práci, aby se zaručilo, že požadovaný produkt bude splňovat očekávání.
- Řešení - Výstupem softwarového vývoje je vyvinout funkční systém řešící určitý problém, který byl definován ve specifikacích.
- Úsilí - Vývoj softwaru je důsledek úsilí, které obvykle zabírá značný čas a námahu na uskutečnění. Pro netriviální řešení je obvykle třeba několik různých lidí se specifickými znalostmi, kteří tvoří tým. Tato oblast se zabývá vším, co je v určitém vztahu k týmu.

2.2.1 Alfy

Alfy jsou subjekty v softwarovém vývoji, kterým chceme rozumět, monitorovat, řídit a kontrolovat až do úspěšného závěru. Mají své stavy a kontrolní seznamy (checklists). Soustřeďují se na dosahování pozitivních výsledků a měli bychom se na ně dívat jako na skupiny, ne jako na individuality. Už jsme mluvili o některých jako příležitost, požadavky nebo tým.

Alfy existují bez jejich konkrétní reprezentace. Existují, i když je nedokumentujeme a bez ohledu na metody vývoje. Jsou to nezbytné věci, které mají tendenci ovlivňovat projekt.



Obrázek 3 Organizace jádra a jeho prvků

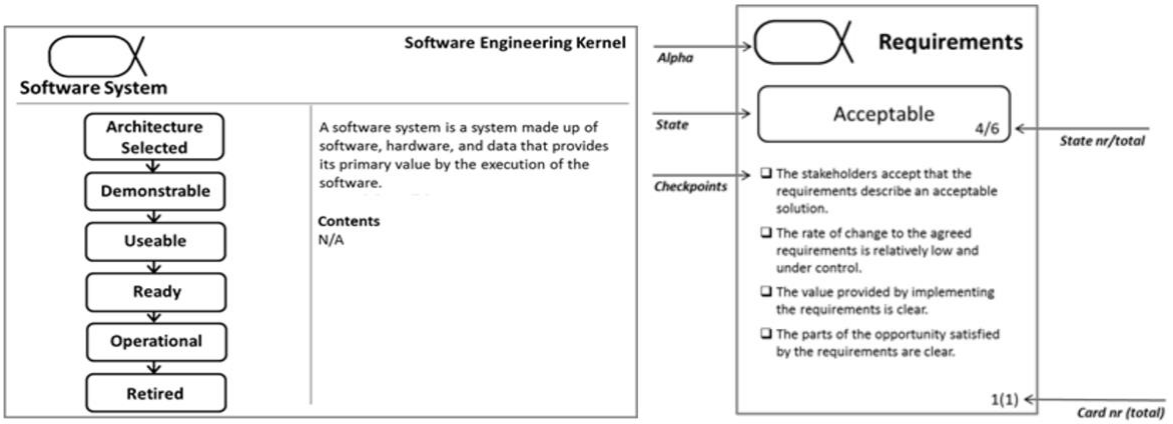
Na obrázku jsou určité alfy, znázorněné vztahy mezi nimi a oblasti zájmů, do kterých patří.

Alfy mohou pomoci k nalezením kořene problému. Avšak pomáhají i při jeho řešení, kdy problém umíme vyjádřit určitým stavem. S každým stavem je spojen předem definovaný kontrolní seznam a je možné se mezi stavy vracet.

Jádro sice ukazuje základy softwarového inženýrství, avšak zejména pro pracovníky, kteří s ním ještě nepracovali, nemusí být snadné ho pochopit. Proto existují cesty, jak se ho naučit používat. Jednou z nejpoužívanějších cest je pomocí karet. Karty usnadňují pochopit Jádro, ale i porozumět, jak ho prakticky využívat s týmem. Karty pro alfy jsou přehledné a snadno použitelné. Dělíme je na karty definování alfy a stavové karty.

Karty jsou jednou z možností, jak vývojářům pomoci pamatovat si Jádro a používat jej. Ze zkušeností víme, že většina týmů potřebuje nějaký mechanismus pro uchovávání stavů Alf a jejich kontrolních seznamů, které budou denně na viditelném místě během jejich aktivit. Když je Jádro aktivní, nevyznačuje to jen věc, kterou je potřeba udělat. Znamená to, co tým aktuálně dělá a obsahuje aktuální pokrok týmu. Jádro dává jednoznačný pohled všem vývojářům, bez ohledu na jejich zkušenosti, kde se v projektu nachází tým a co bude dělat nejdřív.

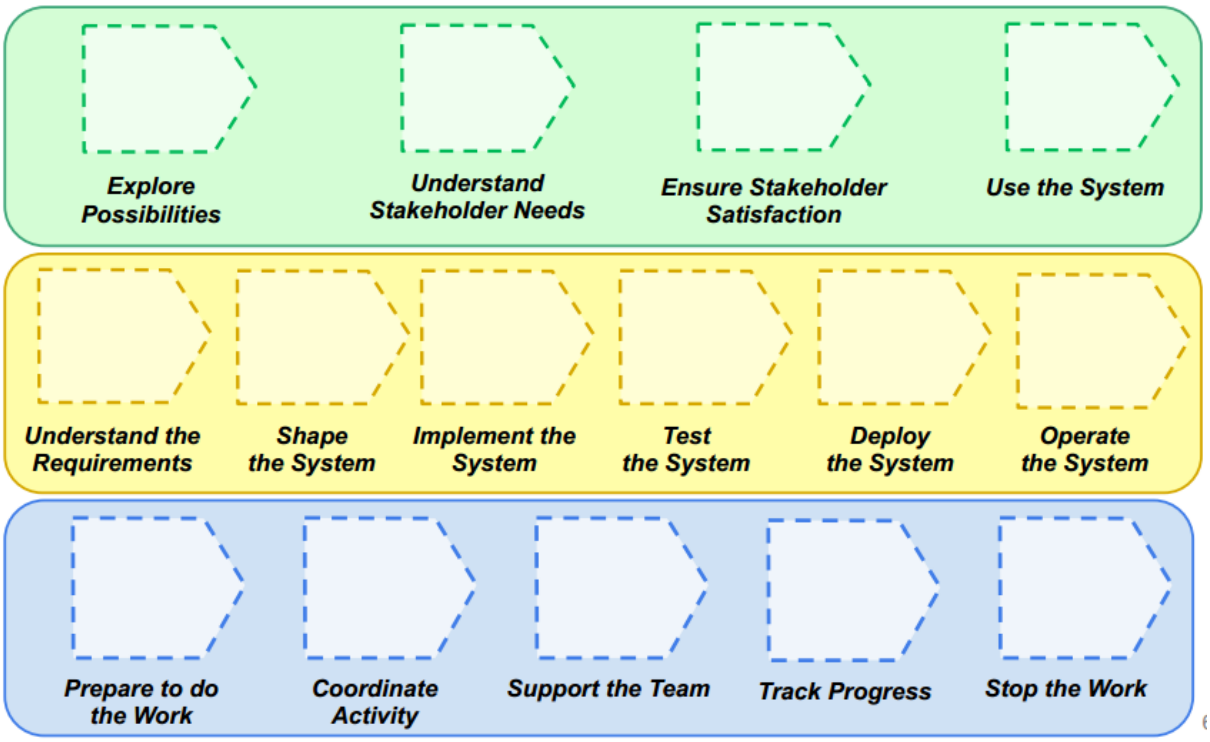
Karty definování alf bývají velké 5x3 palce a sumarizují o čem je dána alfa. Stavové karty bývají ve velikosti vizitek a obsahují konkrétní následující stavy, kde se daná alfa může nacházet.



Obrázek 4 Karta na definování alfy (vlevo) a stavová karta

2.2.2 Místa aktivit (Activity space)

Jádro jako takové nedefinuje žádné konkrétní aktivity, ale definuje určitý počet míst aktivit. Místo aktivity můžeme považovat jako metodologicky nezávislý symbol pro specifické aktivity, které budou přidány později na vrchol jádra.



Obrázek 5 Místa aktivit

Místa aktivit vidíme na obrázku čárkovanými pětiúhelníky a jsou organizovány do specifických oblastí Jádra, jak jsme zmiňovali v předchozí kapitole. Vyjadřují určitou aktivitu, která je potřebná a musí být provedena pro úspěšný vývoj softwaru. Poskytují nám základní informace o problému, před kterými tým stojí. Platí, že místa aktivit čteme zleva doprava a po řádcích. Posloupnost odpovídá tomu, jak jsou jednotlivé aktivity ukončené, ale ne pořadí, ve kterém jsou spuštěny.

2.2.3 Kompetence

Ke spolupráci na vývoji softwarového systému je třeba mít určité schopnosti v různých oblastech. Zejména je nezbytné mít znalosti v příslušné oblasti, v níž pracujete, ale i schopnost porozumět tomu, na čem pracují vaši kolegové v týmu.

Kompetence jsou v Jádře definovány a mohou být zobrazeny v generickém kontejneru pro specifické znalosti. Například programování v Javě není část Jádra, protože tato znalost není základem všech vývojů softwaru.

Běžným problémem při vývoji je nevědomost určité mezery ve svých znalostech, které je třeba mít. Jádro tyto mezery zvýrazňuje a poskytuje rozhraní pro jejich řešení.

Obrázek 6 Kompetence

2.2.4 Metody

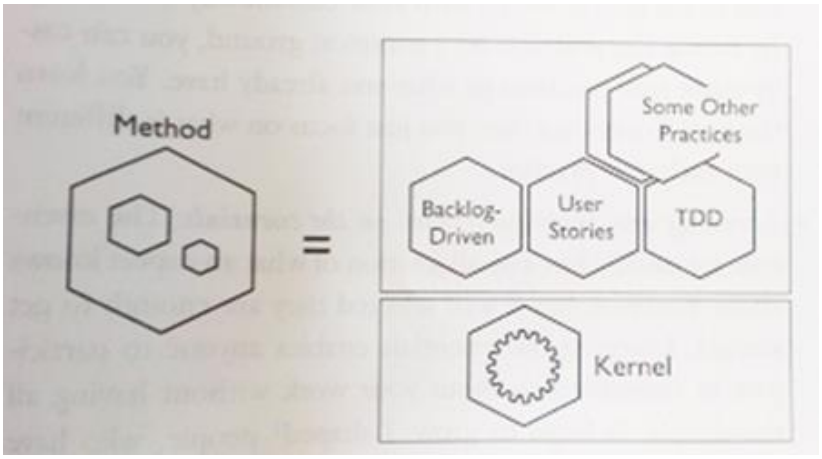
Metoda je kompozice postupů práce, které musí splnit tým pro dosažení svých cílů. Jádro nabízí jednoznačný jazyk a přehled zaměřený na to, co tým produkuje a dělá. Nepoužívají se tradiční dokumenty požadavků, ale uživatelský příběh a z něj vytvořené jednotlivé aktivity. Tyto aktivity jsou vytvořeny pro zlepšení pochopení požadavků, komunikaci se zúčastněnými stranami a obsažení příležitostí zákazníka.

Pracovní postup pomáhá upřesnit, jak by se mělo postupovat při plnění úkolů. Může obsahovat přesný postup jak daný produkt vyvíjet. Přesnost tohoto postupu záleží na dvou hlavních faktorech:

- Znalosti: Zahrnuje znalosti a schopnosti jednotlivců, kterými disponují. Zkušení členové týmu potřebují pouze případné připomínky nebo vzorové příklady. Méně zdatní členové potřebují trénink a naučit se danou problematiku, aby pracovali efektivně.

- Zázemí: Pokud tým pracuje spolu už déle, obvykle mají společné nebo alespoň podobné školení. Takový tým si většinou dobře rozumí a jeho práce nepřináší neshody. Pokud však jde o různorodý tým s různými znalostmi, který je zvyklý používat odlišné metody, je důležitá komunikace a popisy, aby se předešlo nedorozuměním mezi jednotlivci.

Při zahájení využívání Jádra, si tým zvolí sadu nástrojů a postupů, které chce používat. Tato sada postupů je konkrétní metoda. Metoda může být i předdefinovaná, pokud jde o určitý standard nebo již odzkoušený postup řešení dané problematiky. Definovaná metoda je použitelná pro tým jako startovací bod. Metody jsou velmi rozmanité a často jí návrh závisí na unikátní situací.



Obrázek 7 Metoda

Z obrázku je patrné, že metoda je vlastně určitý popis, jak bude tým pracovat a poskytuje pomoc při postupu řešení dané úlohy. Jádro pomáhá strukturovat, co tým dělá a podporuje evoluci projektu.

2.3 Konkrétní výhody

Jako softwaroví profesionálové chceme vytvářet skvělý software rychle a flexibilně. Setkáváme se s většími nástrahami a učíme se množství rozličných technik a domén. Používáme různé postupy a metody. K tomu se musíme neustále vzdělávat a tým jako celek se musí rozvíjet.

2.3.1 Vyvíjet skvělý software

Jádro pomáhá řešit typické problémy, vidět pokrok, zralost toho, co vyvíjíme a komunikovat efektivně. Tradiční metody jsou často statické, jejich popis je těžkopádný a často se neshoduje s tím, co přesně problematika obsahuje. Dva hlavní rysy, které odlišují Jádro od tradičních metod:

- Jádro se soustřeďuje přímo na zlepšování a životnost softwaru - Alfy jsou klíčem k tomu, nepopisují jen metody, ale radí jak dosáhnout zlepšení pokud využíváme metody. Každá alfa má sérii stavů a každý stav má několik kontrolních bodů, které

musí být splněny. Leckdy se při vývoji můžeme podívat, jak jsou konkrétní stavy splněny a pochopit tak podrobně jejich vývoj.

- Jádro má praktickou hodnotu - Pro lepší pochopení, kde se tým nachází, stavy alf pomáhají rychle a jednoznačně určit následující kroky vývoje. Kontrolní seznam definuje, kdy je stav kompletní. Pomocí míst aktivit a aktivit samotných umíme určit pokrok a Jádro nabízí jistý návod jak dosahovat další stavy.

2.3.2 Personální růst

Jádro nabízí prostředí pro sdílení informací mezi softwarovými profesionály a poskytuje pevný základ pro osobní rozvoj. Umožňuje získat zkušenosti, pochopit přidanou hodnotu v týmu, plánovat osobní rozvoj v konkrétních oblastech a učí pracovní postupy. Zkušenosti říkají, že Jádro je aplikovatelné v různých situacích a doménách, ovšem to neznamená, že dokáže řešit všechny problémy. Jednou z nejdůležitějších vlastností je škálovatelnost, protože tím způsobem je možné přizpůsobit systém na unikátní požadavky. Obsahuje 3 dimenze škálování:

- Zvětšení - nabízí doporučení, kdy jsou členové týmu s odlišným školením nebo odlišnými dovednostmi. Doporučení jsou ve formě postupů.
- Natažení - dohoda mezi různými typy vývoje ("in-house", "offshore", "Bespoke", "products"). Každý z nich má různé problémy a potřebuje jiné metody zahrnující jiné postupy.
- Rozšíření - v případě, že požadavků je obrovské množství, více než jeden systém nebo velmi mnoho vývojářů.

2.3.3 Učení

Běžně se lidé učí čtením stránek, knih, chodí na kurzy a čerpají vědomosti od kolegů. Někdy je však matoucí, když se dá informace podat různými způsoby. Jádro nabízí v oblasti učení zejména tyto výhody:

- Papíry, knihy, kurzy jsou sdíleny na jednom místě.
- Autoři mohou znalosti kdykoli inovovat.

Nejlépe se však učí praxí. Jádro se snaží podporovat praktickou dovednost a učení se z ní. Umožňuje měnit týmy a pozice v organizaci podporou nástrojů potřebných při předchozí zkušenosti.

2.3.4 Vývoj

V ideálním případě jsou změny ve způsobu práce evoluční, ale ne revoluční. Obvykle se nezahodí celý postup při práci a nevymění se zcela za nový. Jsou věci, které bychom chtěli zachovat a dělat evoluční změny znamená identifikovat ty nevyhovující části a nahradit je.

3 Využití jádra během iterace

Následující část práce si klade za cíl seznámit čtenáře s tím, jak lze využít prvků jádra pro běh iterace. Zdrojem pro tuto kapitolu práce slouží výhradně kniha (Jacobson, a další, 2013).

Na úvod této kapitoly vysvětlíme některé základní pojmy, s kterými pracujeme v další části práce:

Iterace - je doba určená k vývoji stabilního přírůstku softwarového systému. Délka iterace se běžně pohybuje v rozsahu mezi dvěma až čtyřmi týdny a může zahrnovat celou řadu činností (od získání požadavků až k nasazení výsledného softwarového systému). Softwarový systém typicky roste se zvyšujícím se počtem provedených iterací.

Cíl - cílem každé iterace je vytvoření dalšího stabilního přírůstku softwarového systému. Mimo to je možné specifikovat další cíle, kterých má být v iteraci dosaženo. Příkladem může být demonstrace zcela nové funkcionality nebo zavedení nového způsobu práce. Cíle iterace jsou často svázány s jedním nebo více stavy alf jádra.

Backlog - poté co se tým shodne na tom, co všem má být v iteraci provedeno a stanoví si cíle této iterace, zapíše tyto cíle tzv. backlogu. Backlog může být upřesněn přidáním nezbytných úloh, které je potřeba dokončit, aby byly dosaženy cíle dané iterace.

Úloha - je část práce, kterou lze jednoznačně identifikovat, izolovat a po jejím dokončení schválit jedním nebo více členy týmu. Každý cíl iterace v sobě zahrnuje jednu nebo více úloh.

3.1 Cyklus Plánuj, Proveď, Zkontroluj a Přizpůsob

Iterace je v Essence rozčleněna do čtyř fází, které tvoří opakující se cyklus. Cyklus má fáze Plánuj, Proveď, Zkontroluj, Přizpůsob (Plan, Do, Check, Adapt).

Fáze plánování začíná určením současného stavu. Následně je třeba učit budoucí stav a způsob jak tohoto stavu dosáhneme. Do fáze provádění zahrneme veškeré činnosti vedoucí k dosažení budoucího stavu, mezi tyto činnosti patří i odstranění překážek, které se mohou v průběhu iterace vyskytnout. Ve fázi kontroly jde především o sledování průběhu práce a provádění její kontroly, zda byla skutečně provedena. A nakonec ve fáze přizpůsobování nám odráží, co se skutečně stalo, zkouší najít vhodnější způsoby práce, zlepšit její kvalitu a redukovat ztráty.

Každá iterace se dotýká všech oblastí jádra.

3.2 Plánováni iterace

Těžiště plánování spočívá především v tom správně rozhodnout o tom, co má být v dané iteraci hotovo během následujících dvou až čtyř týdnů. Výsledkem každé iterace je samozřejmě fungující software, nicméně jsou zde i jiné aspekty, na které se musí tým zaměřit. Je třeba si položit otázku, zda tým vyvíjí nejlepší software, jaký mohou a jestli by to náhodou nemohl dělat lépe.

Jádro pomáhá týmu s tím, na co se má konkrétně během vývoje zaměřit.

Při plánování iterace je třeba určit tři věci:

1. Kde jsme? – (tedy současný stav našeho úsilí)
2. Kam směřujeme? – (co dál, jaké jsou cíle pro další iteraci)
3. Jak se tam dostaneme? (stanovení úloh, které je potřeba k dokončení naplánovaných cílů)

Cíle iterace jsou zaneseny do tzv. backlogu společně s úlohami, které vedou k jejich splnění. Backlog je vlastně seznam všeho, co se má udělat.

3.2.1 Plánování pomocí stavů alf

Při plánování iterace nám alfy pomáhají určit, kde jsme a kam směřujeme. Při určování toho, kde jsme, je třeba také pochopit detaily související s technologií, riziky, kvalitou a potřebami akcionářů. Dále je třeba pochopit, kde se nacházíme v rámci celku v našem softwarovém úsilí. Mezi alfami a cíli existuje velmi úzký vztah. (viz. odstavec „Jak zjistit kam jdeme“)

3.2.2 Kde jsme (současný stav)

V rámci Essence existují dvě metody, jak zjistit v jakém stavu se nacházíme.

Realizace pomocí kartiček stavů alf, metoda postupného procházení

1. Do řádku dáme kartičky pro každou alfu a to od počátečního stavu do konečného.
2. Procházíme postupně každým stavem a ptáme se, jestli jsme už dosáhli takového stavu.
3. Pokud jsme stavu již dosáhli, odložíme kartičku vlevo a přecházíme na následující stav, dokud nenalezneme stav, který ještě nebyl dosažen.
4. Odložíme kartičku vpravo společně s ostatními zbývajícími stavy.

Jiný přístup, tzv. poker metoda

1. Všichni členové týmu dostanou kartičky všech stavů dané alfy.
2. Každý člen vybere stav, který si myslí, že nejlépe odráží současný stav úsilí vývoje softwaru.
3. Každý člen dá vybranou kartičkou na stůl textem dolů.
4. Všichni najednou obrátí své kartičky.
5. Pokud všichni vybrali stejnou kartičku, je v týmu shoda.
6. Pokud ne, je nejspíše zavádějící interpretace kontrolních seznamů pro jednotlivé stavy. Tým by měl poté projednat a udělat změny v kontrolních seznamech tak, aby bylo dosaženo shody.

Z toho, že zjistíme, ve kterém stavu jsme, nám plyne i to kam půjdeme, do dalšího stavu alfy.

3.2.3 Kam jdeme (budoucí stav)

Ve skutečnosti mohou být cíle iterace vyjádřeny jako sada cílových stavů alf. Je velmi snadné vybrat, které stavy by měly být vybrány pro další iteraci. Cílové stavy alf, poskytují skvělou podobu cílů, jelikož kontrolní seznam těchto stavů poskytuje jasně definované kritéria pro

dokončení. Vybrání cílových stavů můžeme realizovat kartičkami pomocí metody poker nebo metody postupného procházení.

Do backlogu vstupuje alfa se svým cílovým stavem. Stavy alf nám pomáhají nastavit cíle, které zaneseme do backlogu. K dokončení cílů potřeba splnit řadu úloh, které mohou vycházet z kontrolních seznamů. Z backlogu by mělo být také vidět, které úlohy právě probíhají a které jsou již dokončené. Na úlohách v backlogu pracují během iterace jednotliví členové týmu. Dokončením práce dosáhneme splnění cílů a postupu do dalšího stavu alfy.

3.2.4 Jak se tam dostaneme

Poté co máme stanoveny cíle iterace, je třeba rozhodnout, zda je vůbec možné dosáhnou těchto cílů v rámci časového okna dané iterace. To je typicky stanoveno pomocí úloh, které je nezbytné udělat pro dokončení daného cíle. Stavy alf nám pomáhají tím, že díky kontrolnímu seznamu pro každý stav víme přesně, které úlohy je pro splnění požadovaného cíle třeba dokončit. Samozřejmostí je, že v rámci jednotlivých alf lze prioritizovat, co udělat jako první. Jestli se tým bude věnovat například požadavkům (Requirements) anebo například způsobu práce (Way of Working).

3.2.5 Jak nám jádro pomáhá v plánování iterací

Dobrý plán musí být především kompletní, tedy zahrnovat všechny základní položky a pokrývat celý tým. To je realizováno pomocí alf, prvků jádra, díky nimž máme na paměti různé dimenze softwarového vývoje. Alfy tak pomáhají vytvořit plán, který se dotýká všech dimenzí vyváženým způsobem.

Plán musí být také konkrétní. Tým musí mít prostředek, aby mohl monitorovat pokrok na vývoji proti plánu. Kontrolní seznam pro každý stav alfy nám radí s tím, co je potřeba udělat v dané iteraci. Stejný kontrolní seznam nám také pomáhá určit postup tím, že vidíme, co bylo uděláno v kontrastu s tím, co vše jsme plánovali udělat.

3.3 Provádění a kontrola iterace

Každý den si tým vezme z backlogu úkoly, na kterých se chystá pracovat. Práce pokračuje až do konce iterace, kdy tým vyhodnotí, co se udělalo a určí se tak nový stav úsilí. Cyklus „Plánuj-Dělej-Kontroluj-Přizpůsob“ se opakuje každou iteraci.

Je velmi běžné, že se úkoly zapisují na tabuli, kde jsou kromě „Úkolu“ (To Do) další dva sloupce a to „Dělá se“ (Doing) a „Hotovo“(Done). Na první pohled je tak vidět, na čem se pracuje a co je ještě třeba udělat. Jádro nám nabízí možnost cílové stavy jednotlivých alf interpretovat jako „cíle“(Objectives) iterace, pod které se nám budou sdružovat jednotlivé úkoly. Jednotlivé úkoly si potom tým rozdělí mezi své členy a začne na nich pracovat.

3.3.1 Jak nám jádro pomáhá v provádění a kontrole iterace

Udržet si soustředění na správné úkoly je nezbytné pro dosažení cílů, které byly odsouhlaseny pro danou iteraci. Stavy alf společně s kontrolními seznamy pomáhají týmům, aby si toto soustředění udržely.

Cíle iterace - Tým se shodne na tom, co jsou vlastně cíle iterace. Kontrolní seznamy stavu alf by měly pomoci dohodnout se na tom, co je potřeba udělat nejdřív, tak aby jednotlivé části tvořili smysluplný celek.

Prioritizace úloh - Pokud tým zjistí, že je potřeba dodatečná práce, která nebyla v plánu, mělo by v rámci týmu dojít k přezkoumání kontrolních seznamů cílových stavu alf, které by měly pomoci určit, zda je dodatečná práce něco co by mělo být prioritizováno a nebo něco co může být odloženo.

Připomínkovač chybějících úloh - Každý den se tým podívá na kontrolní seznam požadovaného stavu. Měla by vyvstat otázka: „Máme všechny úkoly, na kterých bychom měli pracovat?“ Odpověď na tuto otázku nám pomůže identifikovat chybějící úlohy.

3.4 Přizpůsobení způsobu práce

Jádro zachycuje podstatu softwarového inženýrství. Připomíná nám, že máme zahrnout akcionáře, že máme myslet na rizika a pomáhá nám soustředit se na nejdůležitější věci. Následující část ukazuje, jak jádro pomáhá upravit způsob práce tak, aby vyhovoval týmu.

Pohledy zpět tzv. retrospektivy jsou skvělým způsobem jak nalézt místa, kde se dá způsob práce s jádrem zlepšit. Typicky lze získat retrospektivu zodpovězením následujících otázek:

- Co šlo dobře?
- Co šlo špatně
- Co může být uděláno lépe?

Způsob práce ovlivňuje všechny členy týmu a každý člen jej může sám ovlivňovat. Toto je další oblast, kde může být jádro užitečné. Tím, že tým mluví o stavech alf v retrospektivách, se může naučit jak některé věci dělat lepé a tím zlepšit způsob práce, který použijí v další iteraci.

3.4.1 Jak nám jádro pomůže v přizpůsobení způsobu práce

Způsob práce udělá jednoznačný. Vývojáři, kteří právě nastoupili, vědí více o programování, než o vývoji softwaru a více o vývoji softwaru, než týmové práci a zlepšování způsobu práce. Protože jsou jejich zkušenosti omezené, potřebují pomoc. Alfy a jejich stavy pomohou týmu uvažovat o způsobu jejich práce a pomáhají jej zlepšit díky provádění retrospektiv. Během retrospektiv mají před sebou členové týmu rozprostřeny stavy alf, aby o nich mohli přemýšlet jako o procesu a nikoliv jako o produktu. Během retrospektiv jsou relevantní jen ty stavy, které náleží do konkrétní iterace. Poté se tým podívá na každý stav zvlášť a pokládá si tyto otázky?

- Co šlo dobře během iterace, dosáhli jsme daného stavu alfy?
- Co šlo špatně během iterace, víme, co nám brání v dosažení daného stavu alfy?
- Co můžeme udělat lépe v příští iteraci, tak aby nám to pomohlo dosáhnout daného stavu alfy?

3.4.2 Provedení změn ve způsobu práce

Každodenní seznamování týmu se stavy alf tak pomůže najít prostor pro drobná zlepšení k přizpůsobení týmového způsobu práce. A pomoci by v tom měly zejména odpovědi na předcházející otázky. Může to rovněž znamenat přidání dalších položek do kontrolních seznamů stavů alf, a to tak, aby to vyhovovalo potřebám týmu. Pokud je to nezbytné tým může definovat i nové, vlastní alfy.

4 Proces vývoje systému[4]

Cílem této kapitoly je popsat styl vývoje konkrétního softwarového systému s pomocí prvků jazyka a pokusit se na příkladech ukázat, jak takový proces probíhá. V příkladu je popisován vývoj mobilní aplikace umožňující díky cloudu průběžné ukládání a sdílení dat na mobilních telefonech.

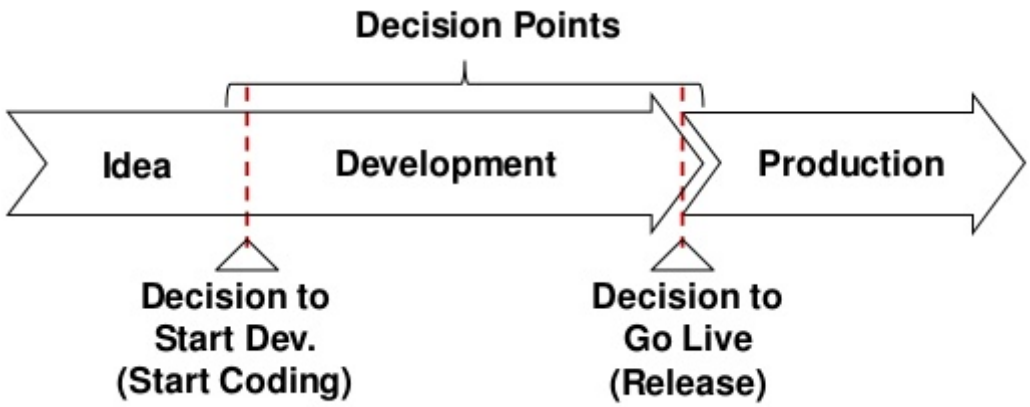
Tým, který se v tomto příkladu zabývá vývojem aplikace, se skládá z několika členů. Angela, představující roli zákazníka, který přišel s nápadem takovou aplikaci vyvinout, nemá žádné technologické znalosti a s vývojem softwaru nemá praktické zkušenosti. V tomto příkladu zodpovídá za definici funkčních požadavků výsledné aplikace.

Dalším členem je Tom, který je zdatným programátorem, řešícím hlavně technologickou stránku projektu.

Na vývoji se kromě Toma dále podílí ještě další dva nově příchozí členové Dick a Harriet. Oba s rozdílnou úrovní znalostí a předchozích zkušeností.

Na celý tým pak dohlíží Smith – vedoucí projektového týmu a Dave, vedoucí celého vývojového oddělení firmy, která se vývojem aplikace bude zabývat.

Před tím, než je sestaven vývojářský tým, dojde k posouzení daného byznys plánu. V případě jeho schválení nastává fáze vývoje, na jejímž konci je další milník, kdy výsledný produkt přechází do produkce. Celý proces je zachycen na obrázku 8.



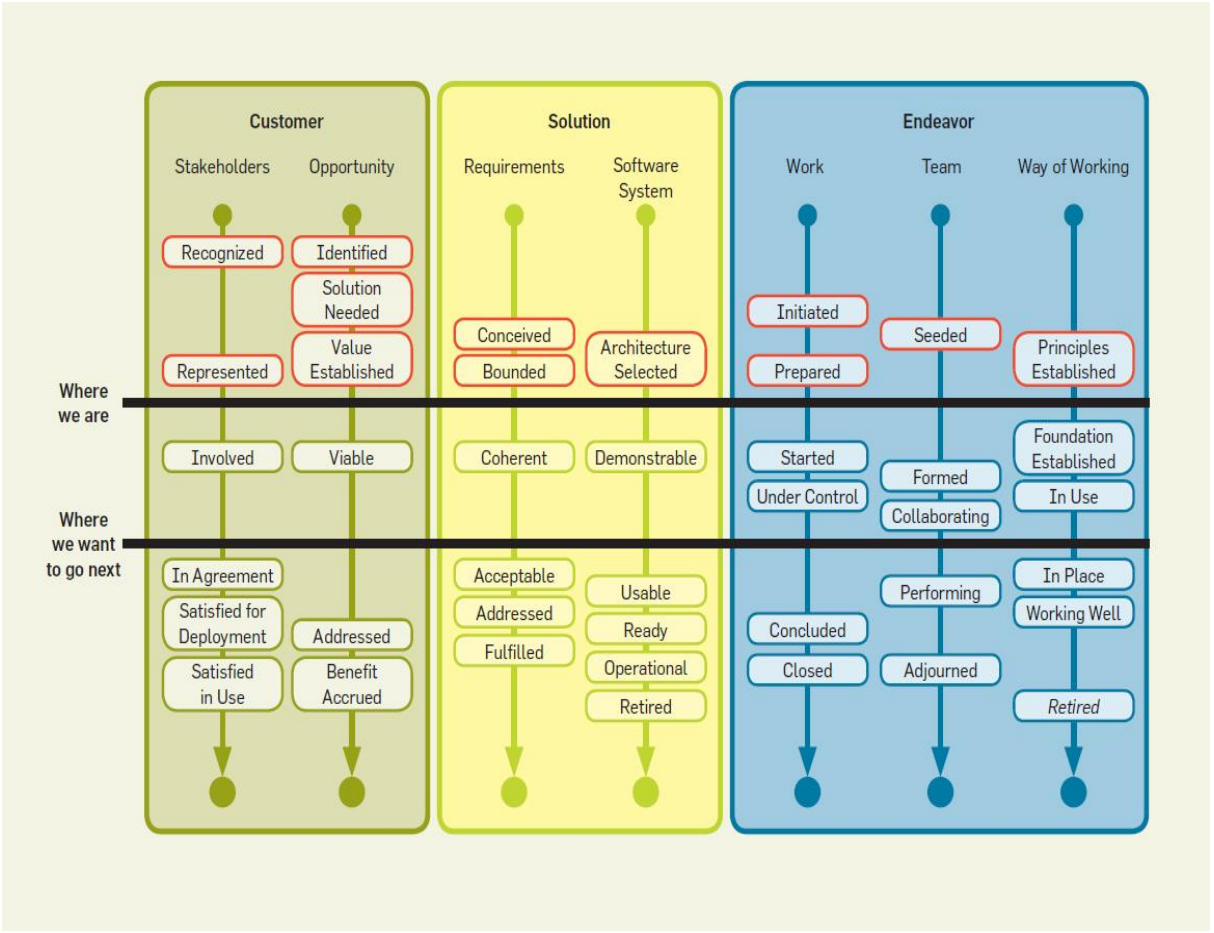
Obrázek 8 Tabulka jádra (kernel table) po úvodní analýze [zdroj: SEMAT]

[4] Zdrojem kapitoly je kniha (Jacobson, a další, 2013)

4.1 Plánování projektu

V úvodní fázi každého projektu je sestaven více či méně detailní popis výsledného produktu včetně definice jednotlivých skupin zákazníků, silných a slabých stránek produktu, analýzy konkurenčního prostředí a dalších charakteristik, které se nazývají byznys plán.

Výsledkem úvodní analýzy je znázorněn na obrázku 9. Ten tvoří odrazový můstek pro další činnosti a fáze projektu.



Obrázek 9 Tabulka jádra po úvodní analýze

Z této úvodní analýzy je možné zjistit, v jakých oblastech se jaká alfa nachází a kde je tedy třeba zapracovat k odstranění nežádoucího stavu. V oblasti „Zákazník“ je tedy nejprve nutné zapracovat, kromě identifikace samotné příležitosti k vytvoření výsledného produktu také na odhalení všech zainteresovaných subjektů.

Tato fáze projektu je často tvořena bez účasti těch členů týmu, kteří následně budou produkt vyvíjet.

Ve fázi „Vývoj“ je pak třeba, aby právě ti členové týmu, kteří jsou zodpovědní za řešení technologické části projektu, pochopili souvislosti celého systému i jeho jednotlivých požadavků a v jejich kontextu pak například navrhli architekturu systému a zhodnotili možnosti používaných technologií.

V oblasti „Úsilí“ je pak třeba zajistit alespoň hrubý odhad časové náročnosti projektu a zároveň navrhnout postup, kterým se bude během dalších fází postupovat. Díky tomu, je možné, aby tým odhadl jednotlivé časové milníky. Prvním z nich je tzv. skinny verze produktu, následovaná použitelnou verzí a celý proces vývoje je zakončen finální verzí aplikace.

4.1.1 Sestavení časového rozvrhu

Na základě seznamu definovaných požadavků je pro tým možné v této fázi sestavit časový plán projektu. Díky identifikaci jednotlivých časových milníků a odhadu, jak dlouho bude trvat dostat se z jednoho k dalšímu, může tým učinit hrubý odhad časové náročnosti celého projektu. Protože je tým rozhodnutý postupovat v iteracích, je možné, aby si sám určil, jak dlouho mu jedna iterace potrvá, případně kolik iterací bude třeba k přechodu na vývoj další verze aplikace.

4.2 Práce na projektu

Tato podkapitola popisuje časovou oblast mezi rozhodnutím pro to sestavit výsledný produkt a jeho uvedením do produkce a to z hlediska různých pohledů, se kterými se tým během tohoto období pravděpodobně setká.

4.2.1 Funkční verze systému

Jako první přichází na řadu budování základního prototypu výsledné aplikace. Ta má jednak za cíl pomoci stakeholderům vyjasnit si veškeré své představy o požadavcích a jednak umožňuje vývojářům ověřit to, co bylo naplánováno v předchozích fázích, a to jak z pohledu technického, tedy použitých technologií a architektury, tak z pohledu procesního, tedy použitých pracovních postupů a stylu řízení.

Z hlediska stakeholderů je důležité, aby byli vývojářskému týmu k dispozici, v případě, že mu bude něco nejasné. Stávají se součástí týmu.

4.2.2 Nasaditelná verze systému

V této fázi projektu, kdy je systém potenciálně připraven k nasazení, je třeba zajistit jeho nasazení do produkčního prostředí a to tak, aby odpovídalo ostatním plánům firmy. Vývojářský tým se musí vypořádat s přáními stakeholderů, kteří chtějí pro první vydání více funkcí, než vyžaduje minimální produkt. Proto se tým rozhoduje, že posledních dvou iteracích se bude snažit těmto přáním vyhovět, avšak nikoliv na úkor testování aplikace.

4.3 Nasazení, podpora a údržba systému

V této kapitole se zaměříme na poslední fázi vývoje systému, která se skládá především z jeho nasazení do produkčního prostředí, jeho podpory a údržby. Dochází zde k předání zodpovědnosti od vývojového týmu směrem k produkčnímu týmu a to při splnění zejména těchto požadavků:

- Stakeholdeři jsou s výsledným produktem spokojení
- Příležitosti a požadavky byly naplněny
- Software je otestovaný a připravený na nasazení

Vývojový tým nicméně stále musí být dostupný, protože dokud nedojde k úspěšnému nasazení, vývojová práce na projektu není dokončena.

Produkční tým řízený Jonesem je zodpovědný za nasazení všeho softwaru vyprodukovaného vývojovým týmem a také za jeho podporu. Podpora je omezena na opravování kritických chyb nebo změn v konfiguraci. Všechno ostatní (zejména návrhy na vylepšení) jsou předány zpět vývojovému týmu.

4.3.1 Okamžik nasazení („going live“ milestone)

Oba týmu jsou se stakeholdery domluveny na datu nasazení aplikace a také na tom, že během první čtyřech týdnů bude aplikace v takzvané záruční době, kdy bude produkt testován především „early adoptery“ a budou sledovány všechny ostatní okolnosti, které před finálním nasazením nebylo možné otestovat. Pokud vše půjde dobře, vývojovým tým pozbude veškerou zodpovědnost za projekt a ta bude přenesena již pouze na produkční tým.

V oblasti „Zákazník“ jde zejména o získání zpětné vazby od stakeholderů, kde se důraz zaměřuje především na to, zda byly naplněny původní cíle a jestli přínos aplikace naplňuje příležitosti, jejich naplnění bylo hlavním cílem vývoje celé aplikace

V oblasti „Vývoj“ jde pouze o udržení systému v provozuschopném stavu a sledování, zda-li jsou požadavky naplněny i v produkčním prostředí.

Poslední oblast „Úsilí“ se zaměřuje na předání zodpovědnosti směrem od vývojového k produkčnímu týmu. V tomto momentě života aplikace se rovněž očekává, že spolupráce obou týmu na svém maximu.

4.3.2 Podpora systému

V této fázi již došlo ke kompletnímu předání odpovědnosti z vývojového na produkční tým a tento tým má tak plnou zodpovědnost za produkt. O produkt se bude starat až do doby než bude, jeho podpora ukončena.

Oblast „Zákazník“ představuje zejména sledování míry spokojenosti zákazníků a velikosti získaného přínosu. Sledování těchto dvou ukazatelů umožní stakeholderům rozhodnout, zda-li a jak dlouho má být produkt podporován.

V oblasti „Vývoj“ probíhá pouze standardní podpora tak jak byla popsána v předchozích kapitolách.

Poslední oblast „Úsilí“ již také nepřináší mnoho nového, je nutné sledovat stav jejich práce, týmu a způsobu jakým pracují.

5 Škálovatelnost vývoje pomocí Jádra[5]

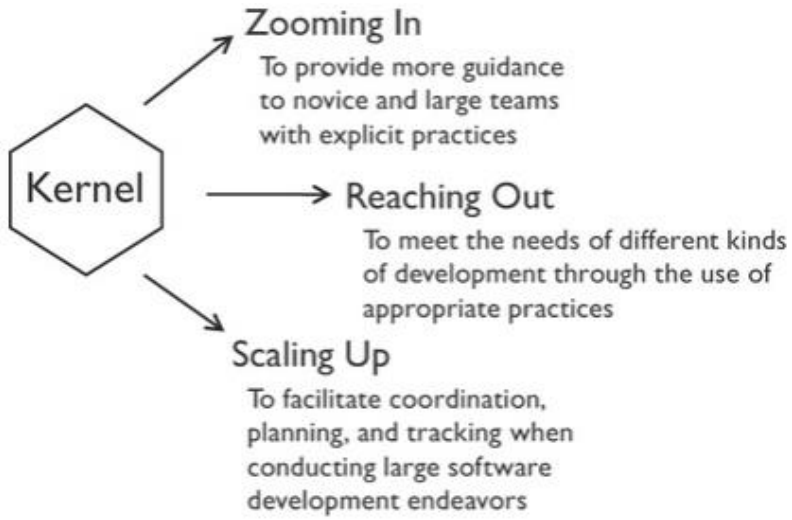
V předchozích kapitolách bylo popsáno, jak může tým použít jádro pro vlastní vývoj softwaru. Předpokladem bylo, že členové týmu jsou kompetentní a zvládnou rozhodnout, co dál dělat podle stavů alf. Ale půjde toto použít i pro ostatní případy a druhy vývoje různých zkušeností a velikostí vývojových týmů? Tato kapitola se tedy zaměří na to, jak moc je vývoj za pomocí „Jádra“ škálovatelný dle různých dimenzí a jak mohou praktiky postavené na jádru pomoci s vypořádáním se s různými výzvami problémy komplexního vývoje.

5.1 Škálovatelnost a dimenze škálovatelnosti

Jádro obsahuje pouze podstatné prvky softwarového inženýrství, ale neobsahuje vše, co tým potřebuje vědět a co potřebuje dělat ve specifických situacích. Je tedy dobré škálovat veškeré snahy podle těchto specifických situací. V případě většího vývoje lze většinou narazit na 3 hlavní výzvy, které je třeba překonat.

- Ne všichni členové týmu jsou dostatečně kompetentní. Někteří potřebují jednoznačnější popis nad rámec toho, co poskytuje Jádro.
- Různé druhy vývoje mají různá rizika a omezení a proto čelí jiným výzvám. Jaký je tedy nejvhodnější přístup k vývoji
- Při velkém vývoji se z koordinace práce mezi členy týmů stává výzva. Jak tedy zajistit aby členové týmu mezi sebou diskutovali, sdíleli a dohadovali se o svých plánech?

Toto jsou hlavní výzvy pro škálovatelnost. Při posuzování co „škálování“ znamená je tedy potřeba vzít v úvahu 3 odpovídající dimenze škálovatelnosti, které mohou nastat najednou ve reálných situacích.



Obrázek 10 Dimenze škálovatelnosti

- **Přiblížení**: První dimenze je o poskytování rad nad rámec jádra. Je to důležité především v případech, kdy jsou v týmu někteří nezkušení členové, co potřebují větší vedení. Také je to nutné u velkých týmů obsahující členy různých znalostí a

[5] Zdrojem kapitoly je kniha (Jacobson, a další, 2013)

zkušeností, protože je důležité zajistit pokračování vývoje, tak aby si všichni rozuměli. Toto dodatečné vedení jádro SEMATu popisuje ve formě praktik, které jsou rozšířením tohoto jádra. SEMAT tedy poskytuje cestu jak systematicky popsat a sestavit tyto praktiky. Pro členy týmu je to pak jednoduchá cesta k pochopení jak tyto praktiky aplikovat a jak spolu různé praktiky souvisí.

- **Rozšíření**: Druhá dimenze je o rozšiřování „jádra“ vhodnými praktikami pro specifické potřeby různých druhů vývoje. Každý druh vývoje je jiný. In-house vývoj je odlišný od outsourcovaného, vývoj webové aplikace je odlišný od vývoje lokálních aplikací. Každý druh vývoje má vlastní sadu výzev, problémů a požadavků vyžadující vlastní sadu praktik. Dimenze rozšiřování je tedy o výběru správných praktik a metod pro vyřešení všech problémů vlastního vývoje. Týmy si tedy pomocí praktik tvoří vlastní metody nad rámec „jádra“. Velké organizace mají obvykle množství vlastních před-vytvořených metod na výběr. Každá před-vytvořená metoda představuje souhrn povinných praktik nad jádrem, aby měly týmy pokyny a postupy jak začít.
- **Škálování**: Třetí rozměr je o rozšíření použití jádra od týmů s malým množstvím členů a menší komplexitou k vývoji zahrnujícím velké týmy a vysoce komplexní produkty (např. firemních systémů). Tyto komplexní systémy jsou velice komplexní a obvykle zahrnují hodně různých frameworků a technologií a také jsou vyvíjeny různými způsoby. Tím pádem se již nezabýváme jednoduchým softwarem jednoho týmu, ale komplexním systém s velkým množstvím sad požadavků, množstvím podsystémů vyvíjených větším množství týmů. V takových situacích je důležité mít nástroje a mechanismy jak dobře zorganizovat práci a týmy, jak znázornit postup a jak zkoordinovat

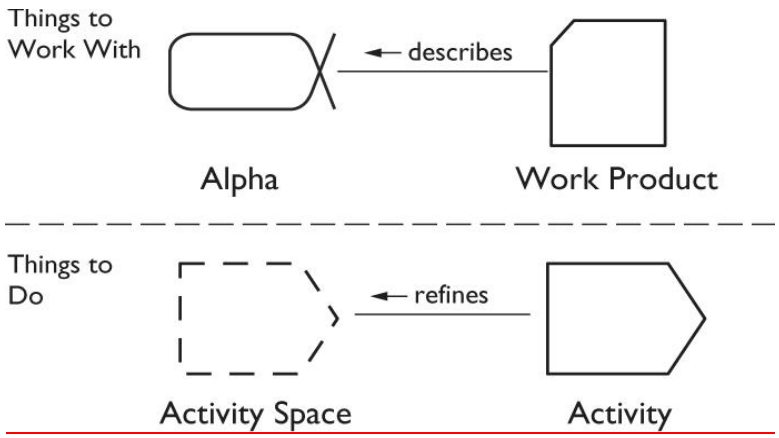
V těchto komplexních situacích je důležité poskytnout nástroje a mechanismy samosprávy pro týmy. Jádro s alfami poskytuje nástroje pro týmy pro navržení cesty jak spolupracovat v rámci, tak i mezi týmy. Tradiční přístupy škálovatelnosti často spoléhají na posílení hierarchie a přidání více kontrol, z čehož může vzniknout mnoho problémů a nežádoucích konfliktů. Moderní přístupy zdůrazňují samosprávu, samo organizaci a samořízení. V další části demonstrována škálovatelnost jádra za použití zmíněných tří rozměrů škálovatelnosti.

5.2 Přiblížení pro poskytnutí detailů (Zooming in to provide details)

Jádro se nesnaží obsáhnout všechny možné návody, pouze ty podstatné. Ale pro nezkušené členy týmu jsou potřeba podrobnější pokyny. V rámci větších organizací je přibližování taky důležité protože členové týmů mají různé zkušenosti a přiblížení je důležité proto, aby si všichni správně rozuměli. SEMAT tedy poskytuje návod na systematický popis a sestavení praktik. Praktika tedy poskytuje návod na řešení některých situací ve vývoji, které jsou třeba přiblížit. Například když má tým problémy s prováděním akceptačních testů, tak potřebují popsat praktiku akceptačního testování. Mimo pochopení jak praktiky používat je také důležité pochopit jak používat a kombinovat různé praktiky.

Jádro poskytuje přesný popis praktik, jak je vidět na obrázku 11. Popisuje to jednoduchým stylem pro snazší pochopení i pro nezkušené členy. Část „S čím pracovat“(Things to work

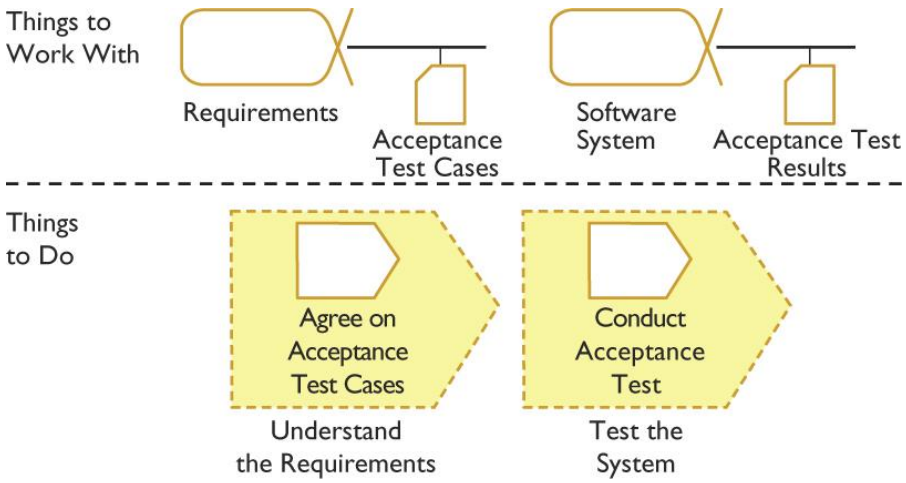
with) obsahuje alfy, o kterých se mluví v předchozích kapitolách, a produkty práce, což jsou různé dokumenty, které tým tvoří při práci s alfami. Část „Co udělat“(Things to do) obsahuj činnosti a mezery činností, které popisují jak přecházet mezi stavy alf.



Obrázek 11 Obecná praktika podle jádra

5.2.1 Příklad: Praktika Akceptačního testování

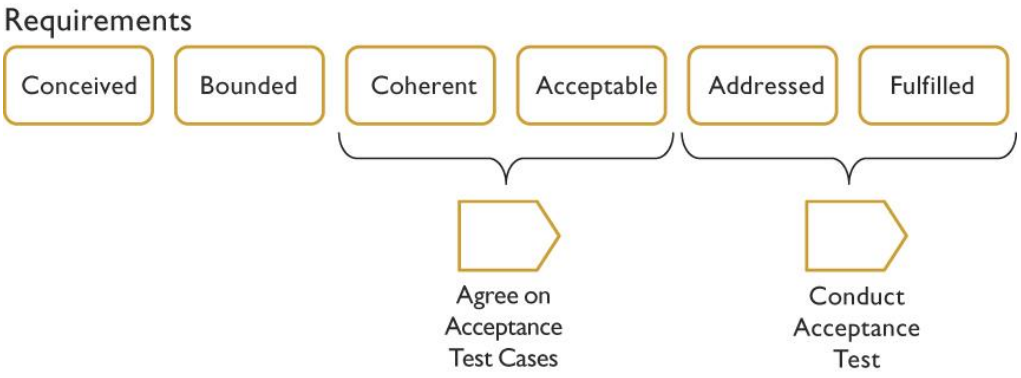
Cílem této praktiky je dohodnutí se na testovacích případech na začátku vývoje z důvodu jejich použití pro řízení vývojových prací.



Obrázek 12 Praktika akceptačního testování

Tato praktika v části „**S čím pracovat**„ obsahuje 2 alfy - alfu Požadavků a alfu SW systému. Z výše uvedeného obrázku lze vyčíst, že požadavky se objasňují pomocí dohodnutí se na testovacích případech a že kvalita systému se vyhodnocuje podle výsledku akceptačních testů. Část „**Co udělat**“ zobrazuje návody k provedení činností. Jmenovitě že dohodnutí se na schválení testovacích případů je cesta k pochopení požadavků a že je při provedení akceptačních testů se provede test systému.

Obrázek 13 zobrazuje použití výše popsané praktiky a tedy stavy vybrané alfy a ukazuje činnosti, které je potřeba udělat pro přechod mezi stavy. V našem příkladu viz obrázek níže.



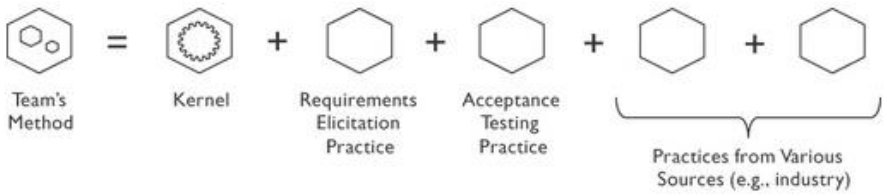
Obrázek 13 Použití praktiky akceptačního testování

SEMAT poskytuje jednoduchý jazyk pro popis praktik, které jsou v jádru specifikované. Začínat od jádra je důležité, protože jádro poskytuje terminologii a jednoduchou cestu pomocí alf jak postupovat ve vývoji a také poskytuje systematický přístup jak popisovat a sestavovat praktiky. Jádro také obsahuje návod jak spojovat praktiky do složených praktik.

5.3 Rozšíření na různé druhy vývoje (Reaching out to different kinds of development)

Různé vývojové snahy (endeavor) jako např. vlastní vývoj, zakázkový vývoj či outsourcing) čelí různým rizikům a mohou narazit na různé problémy. Dimenze rozšiřování je tedy o vybírání správných sad praktik pro vývojový tým. Sada těchto praktik se nazývá metoda. Protože metody přímo ovlivňují práci týmu, snaží se Essence postrčit tým ke tvorbě vlastních metod místo využívání přikázaných cizích metod.

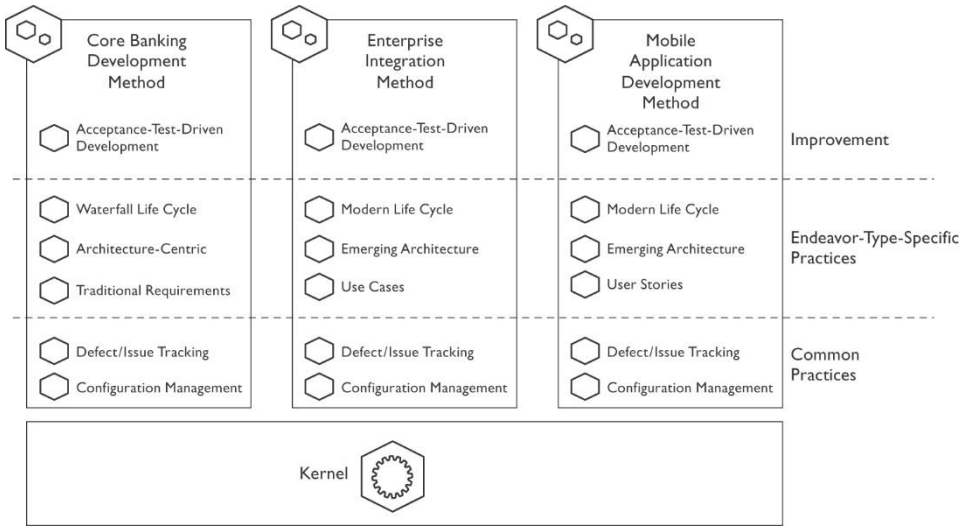
Obrázek 14 popisuje přístup jádra ke tvorbě metod. Ty při tvorbě vychází z jádra, dále si vybírá některé vlastní praktiky a také potřebné ověřené praktiky (best practices) z oboru.



Obrázek 14 Sestavení metody

Essence doporučuje tvořit praktiky a vlastní metody v průběhu celého životního cyklu vývoje, protože různé metody mohou do vývoje přinést nové alfy. Např. praktika testování přinese alfu test, praktika Scrummu přinese alfu iterace a alfu sprintu apod.

Ve velkých organizacích lze běžně potkat mnohá vývojová úsilí mnohého druhu. Například banka může usilovat, jak o zlepšení vlastních bankovních systémů, tako vývoj mobilní aplikace. Na níže uvedeném obrázku 15 je uveden příklad architektury metod v bance. Jsou zde vidět jednotlivé metody postavené nad jádrem a jejich praktiky zařazeny do různých kategorií. Obecné praktiky, které lze použít vždy, praktiky specifické pro danou snahu a praktiky zaměřené na snahy o zlepšení.



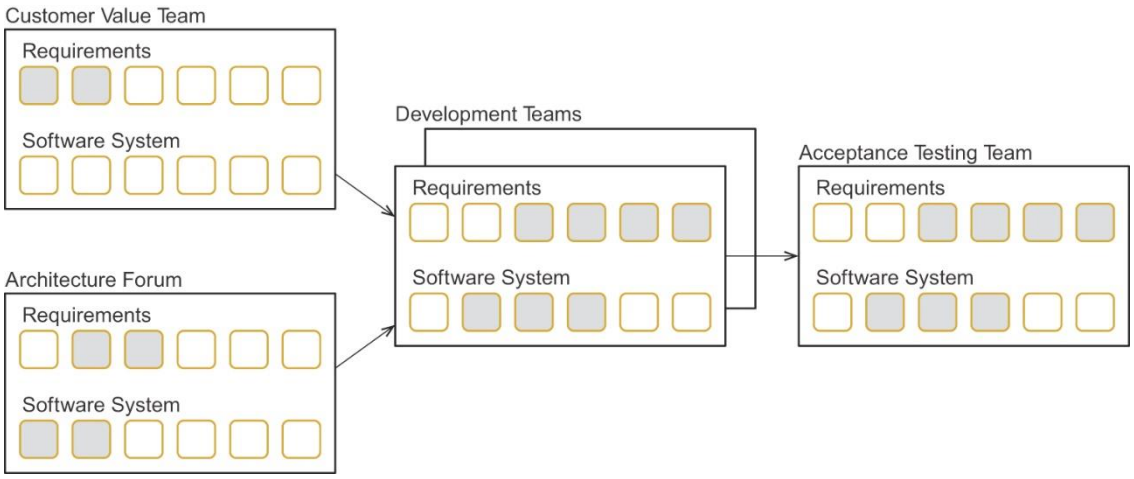
Obrázek 15 Příklad architektury metod v bance

Ve velkých organizacích je zcela běžné, že lidé, co rozhodují o použitých metodách, nejsou ti, kdo tyto metody používají. Essence zdůrazňuje důležitost výběru metod přímo týmem, který je používá a jádro poskytuje prostředky pro popis, přizpůsobení a sestavení vlastních praktik a metod přímo pro tým, který se na nich přímo dohodne místo, aby jim to přikázal někdo svrchu.

5.4 Škálování až k rozsáhlému a komplexnímu vývoji (Scaling up to Large and Complex development)

Třetí dimenzí je škálování z jednotlivých vývojových týmů, k vývoji obsahující několik systémů, několik požadavků a několik týmů, které spolu musejí komunikovat. Koordinace a komunikace při takto velkém vývoji je často hodně náročná a vyžaduje hodně plánování (rozsah, priority, čas, rozpočet), zajištění kvality a mnohem více. Jádro poskytuje nástroje spolupráce jak v rámci tak mezi týmy. Jádro pomáhá při tvorbě týmů, při vizualizaci postupu a s koordinací vývoje s více týmy.

Essence dává přednost agilním přístupům spolupráce než tradičním. Např. dává přednost Fórům spolupráce (Collaborative Forums) před tradiční „Top-down“ hierarchií. Na obrázku 16 je znázorněno zaměření jednotlivých týmů v rámci vývoje.

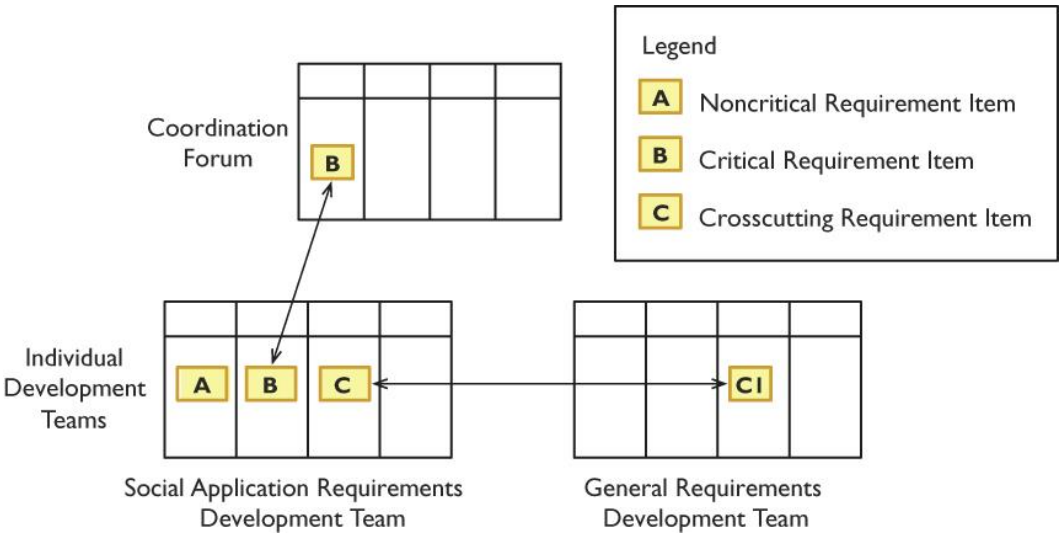


Obrázek 16 Zaměření týmů znázorněné jednotlivými stavy alf

V každém obdélníku jsou vypsány alfy a jejich stavy , šedý stav znázorňuje ten stav na který se jednotlivý tým zaměřuje a v které části vývoje se tedy angažuje. Znázorňuje to tedy přehledně, který tým je za co zodpovědný a na co se má zaměřit.

Ve velkých organizacích je docela jednoduché, když je špatné vedení, aby jednotlivý tým ztratil přehled o celkových prioritách projektu a dělal si své. Proto Essence doporučuje tzv. Fóra spolupráce, která by koordinovala práci jednotlivých týmů a kontrolovala, zda se jednotlivé týmy nad něčím zbytečně nezdržují.

V případě velkých týmů je potřeba týmy správně koordinovat práci a nějak ji znázornit pro členy týmů. K tomu slouží úkolové tabule (task board) viz obrázek 17.



Obrázek 17 Vizualizace spolupráce mezi týmy pomocí úkolové tabule

Obrázek 17 ukazuje příklad úkolové tabule obsahující různé položky požadavku, různých důležitostí rozdělené mezi jednotlivé týmy. Nejjednodušší je **nekritická položka**, kterou jednotlivý tým zvládne sám bez pomoci ostatních týmů. **Kritická položka** je důležitá pro další vývoj je tedy užitečné její vývoj probrat s koordinačním fórem např. exportu do pdf u kancelářské aplikace. Průřezové položky jsou komplexní a nejdou tedy přiřadit pouze

jednomu týmu. Např. při vývoji spisovny se na samotný vývoj soustředí jeden tým a druhý se postará o tvorbu databáze k této spisovně.

Tradiční přístupy k vývoji mohou u velkých vývojových projektů tíhnout k řízení projektu, bez ohledu na to, s čím se potýkají nižší články, což může vyústit ke špatné koordinaci, špatnému porozumění při vývoji a nečekaný zdržením. Essence se snaží jít spíše cestou samo-organizace a samosprávy, kde si jednotlivé týmy mohou promluvit do samotného vývoje a směrovat ho, což jim zlepšuje motivaci a celý vývoj je tím pádem výsledek spolupráce jednoho velkého týmu místo dílčích výsledků, které zadal vedoucí. Alfy jádra jsou tedy skvělým nástrojem pro organizaci práce, zobrazování postupu a koordinace práce mezi týmy.

6 Jak se mění práce s metodami

V předcházejících částích bylo popsáno, jak může tým jádro použít. Následující část se bude zabývat metodami, a jak se za pomoci jádra mění práce s nimi. Každý tým potřebuje specifické metody, ale díky jádru, nemusí začínat z ničeho. Jádro může být pro tým jako odrazový můstek. (Jacobson, a další, 2013)

Při vývoji se členové týmu zabývají metodami a hodně o nich přemýšlejí, i když ne přesně vědí co, která metoda znamená. Je potřebné dát týmu možnost vytvářet si své metody nebo stávající metody upravovat, když si to situace vyžádá. Samozřejmě to nesmí být na úkor kvality výsledného produktu. Metoda musí být ve vlastnictví řešitelského týmu a tým musí být agilní, aby ji dokázal co nejjednodušeji upravit. (Jacobson, a další, 2013)

6.1 Přemýšlení o metodách

Vývojář má za úlohu dokončit svoji práci v co nejvyšší kvalitě, na čas a v rozpočtu stanoveném zákazníkem. Cestou k cíli však vývojáře čekají různé výzvy, jako např.:

- Získaní skutečných požadavků od klienta.
- Implementovat požadavky klienta správně.
- Vybrat nejvhodnější technologií.
- Spolupráce v rámci týmu.
- Zabezpečit kvalitu systému a jeho architektury.
- Dodat systém zákazníkovi v čas.

Výzev, kterým musí vývojář čelit, je určitě více, a každý vývojář může na dané výzvy reagovat odlišně. Když vývojář přemýšlí o jednotlivých výzvách, přemýšlí vlastně o metodách, které by v řešení aplikoval a to i bez toho, že by si byl toho vědom.

Metoda se musí přizpůsobit v daném kontextu. Jiné metody se použijí při práci v týmu a jiné při získávaní požadavků od klienta. Pro jeden tým mohou být uživatelské příběhy (user stories) postačující, v jiném týmu, který se zabývá vývojem specializovaných systémů (armáda) to může být nevyhovující nebo nedostačující metoda. Vybraná metoda by měla, přinést úspěch a to tím, že pomůže vývojářům vytvořit kvalitní systém a v neposlední řade uspokojí potřeby a požadavky klienta.

Diskuse nad výběrem metody může být různá. Záleží na zkušenostech jednotlivých členů týmu a také jak velké rozdíly mezi nimi jsou. Např. čerstvý absolvent bude přemýšlet méně nebo vůbec nad tím, jak pracuje nebo jaké metody při práci používá, na rozdíl od zkušeného vývojáře.

Proto je možné jádro použít jako základní kámen, na kterém začne tým stavět a sdílet metody používané v projektu v rámci týmu. Zrychlí dohodu o používaných metodách v rámci týmu a také urychlí hledání odpovědí na možné problémy v projektu. Jádro tak práci s metodami ulehčí, udělá ji přirozenější a tým může zlepšovat své metody bez toho, aby o tom vůbec přemýšlel.

6.2 Agilní práce s metodami

Při práci s metodami je potřebné být agilní. Neznamená to však, že tým může použít jenom agilní metody. Jde zejména o jednoduchost, jakou sestavujeme nebo přizpůsobujeme metodu. Je to potřebné hlavně kvůli tomu, že při vývoji softwaru se tým učí stále nové věci. Díky získaným zkušenostem pak tým může zlepšit svoji práci a také používanou metodu, která se tak bude vyvíjet, nebude fixní.

Agilní vlna, která přišla začátkem tohohle tisíciletí, zdůraznila důležitost lidí a zainteresovanost zákazníka na řešení projektu. Tato vlna přinesla změnu a vyžádala si také změnu myšlení. Podobnu změnu je potřeba udělat také u metod, když o nich přemýšlíme. Změny vycházejí z Agilního manifestu[6]:

- Celý tým je vlastníkem metody, ne jen pár jednotlivců.
- Zaměřit se na použití metod a ne na jejich popis.
- Metodu týmu rozvíjet, než ji nechat neměnnou.

6.2.1 Celý tým je vlastníkem metody, ne jednotlivci

Tradičně bývá dichotomie mezi tvůrci metod a uživateli metod (např. vývojáři).

- tvůrce metodu definuje, ale už ji tak často nepoužívá
- uživatel, který získá reálné zkušenosti při práci s metodami, už není tvůrcem dotazován nebo nemá čas na poskytnutí zpětné vazby a případné vylepšení.

Jádro se snaží tyto rozdíly odstranit a poskytnut tak rovnováhu pro všechny zúčastněné strany. Zdůrazňuje tak hodnotu každého člena týmu a určuje, který způsob práce je pro tým optimální. Alfy, stavy a checklisty dávají velmi jednoduchý, ale za to silný nástroj za pomoci kterého může tým využit vlastnictví metody.

6.2.2 Zaměřit se na použití metod a ne na jejich popis

Když se např. vývojáři baví o metodách, které používají, tak se baví v rovině popisu. Mýt popis k metodě je správné, protože to umožní novým, ale také existujícím členům týmu se lépe seznámit s metodou. Často se však z těchto popisů vynechají informace o tom, jak se s danou metodou každodenně pracuje. Popis k metodám bývá také těžkopádný. Proto je potřebné s metodou pracovat a dále ji rozvíjet, čemu se věnuje následující část.

6.2.3 Metodu týmu rozvíjet než ji nenechat ji neměnnou

U metod neexistuje univerzální řešení, které by bylo možné použít na kterýkoli projekt. Z toho vyplývá:

- Není možné sebrat kteroukoli metodu a bezhlavě ji následovat. Každá metoda se musí přizpůsobit dané situaci.
- Když se metoda přizpůsobí, tak po čase se tým dozví nové skutečnosti a je potřebné metodu opět přizpůsobit na novou situaci (v důsledku nových skutečnosti).

[6] Více informací o Agilním manifestu na http://agilemanifesto.org/principles.html

Metoda není nikdy fixní. Tým musí usilovat o vývoj metody, dokud není ukončena práce na projektu. Z toho vyplývá:

- Vždy být připraven na nový a lepší způsob práce.
- Vždy brát do úvahy aktuální stav vývoje projektu při úvaze o změně (vylepšení) metody.

Jádro ulehčí rozvoj metod. Stačí začít s jádrem a postupně přidávat metody, které už tým zná, případně nahradit nevyhovující novými.

7 Co nového metodika přináší[7]

Autoři SEMATu a jádra, nezačínali na zelené louce. Softwarová komunita se snaží zkoumat práci s metodami už více než 50 let. Autoři tvrdí, že pokusy v minulosti nebyly moc úspěšné, protože nedokázali oslovit velké množství profesionálů v oblasti softwaru a jeho vývoje.

V následujících částech jsou popsány nové myšlenky, resp. metody, které metodika přináší.

7.1 Obnova metod

S velkým počtem vývojářů ve světe existuje také velké množství metod používaných při práci. Problém je možné vidět v tom, že každá z těchto metod je vlastně izolována od ostatních a nesdílejí společné základy (jádro). Problém teda není v tom, že by bylo málo metod nebo neexistovaly kvalitní metody. Problém je v tom, že vývojáři většinou nepracují s metodami efektivně, když je chtějí přizpůsobit nebo když se chtějí inspirovat u jiných týmů.

SEMAT našel společné základy, které jsou obsaženy v každé metodě. Ať už se jedná o psaní kódu, osvojování metod, nebo návrh architektury, vždy tam existuje společný základ, jádro. Najít ty správné elementy je proto klíčové. SEMAT ulehčí hledání těchto prvků a také zaručí, že na těchto prvcích se shodla větší komunita lidí, než len lidi v rámci týmu, vzniká tak standard. SEMAT tak dělá jádro lehce upravitelné, praktické a rozšiřitelné.

Cílem jádra je udělat metody více praktické a intuitivní, aby ulehčily práci softwarovým profesionálům. To znamená:

- Více dbát na použití metod než na jejich popis.
- Dbát na to, co pomáhá softwarovým profesionálům (uživatelům metod) než to, co definovali tvůrci metod – „Softwaroví profesionálové jsou králi, tvůrci metod jsou rytíři, kteří slouží králům.“ (Jacobson, a další, 2013)
- Konkrétní, intuitivní a grafická syntaxe před formální a komplexními opisy. Příkladem jsou alfa karty a stavy.

Jádro je měnitelné. Jádro poskytuje předdefinované sady slov souvisejících s vývojem softwaru, ale jádro není slovník, určený jenom pro čtení. Jádro je dynamické a s tím, jak se vývoj posouvá kupředu, se stejně vyvíjí také jádro a jeho jednotlivé prvky – alfy přecházejí ze stavu do stavu. Předdefinované checklisty je možné rozšířit o své vlastní požadavky.

Jádro je také rozšiřitelné. Obsahuje základní prvky, které se nacházejí ve všech projektech vývoje softwaru. Je možné přidávat různé metody nebo zkušenosti do jádra, tj. např. podporu pro uživatelské příběhy (user stories), párové programování, atd. Nad jádrem tak vznikají nové způsoby práce a tým tak může skládat a vytvářet nové metody potřebné k práci. Tak vznikají různé metody, které mají společný základ, kterým je jádro.

7.2 Oddělení zájmů v metodikách

SEMAT přistupuje k metodám odlišně, než se přistupovalo v minulosti. Tento přístup je možné nazvat jako „oddělení zájmů (separation of concerns)“ ve vývoji také označováno

[7] Zdrojem kapitoly je kniha (Jacobson, a další, 2013) a (Jacobson, a další, 2014)

zkratkou SoC. Jedná se o princip, kdy se systém navrhne tak, že je vytvořený z modulů, z kterých se každý stará o svoji část. Nevzniká tak špagetový způsob vývoje. Systém je tak lehce upravitelný a udržovatelný. Tento způsob je možné použit také u metod a to:

- Specifikujeme jádro.
- Vytvoříme rozšíření bez toho, abychom komplikovali nebo měnili specifikaci jádra.

Tím, že se nebude měnit jádro, tak tým může počítat s tím, že zásady stanovené v jádru budou platné také po přidání rozšíření do jádra.

Zásada oddělení zájmů platí také pro postupy. Za pomoci SoC je možné vzít postup z jedné metodiky a spojit ji nezávisle s druhým postupem z jiné metodiky. Takhle postupovali také autoři Essence. Tým tak může používat různé metody z různých metodik. Jádro mají společné, tým tak má pevný základ, společný jazyk a způsob spolupráce. Tímto způsobem se také metody rozvíjejí.

Dále je možné oddělit alfy od dokumentace nebo popisu práce. Tradiční postup je takový, že metody kladou velký důraz na dokumentaci jako nástroj, kterým je možné měřit průběh projektu. Agilní přístup tohle tradiční myšlení pozměnil a dává přednost fungujícímu softwaru před hromadou dokumentace. Fungující software však nemusí být dostatečným nástrojem na měření průběhu projektu. Možná byly nedostatečně definované požadavky nebo je software těžko modifikovatelný nebo testovatelný. Proto potřebuje tým měřit svůj průběh jiným způsobem, skrz jinou dimenzi jako např. požadavky, práce, tým atd. Tu je možné použít alfy a za pomoci jejích stavů měřit průběh. Protože dokumentace může být tvořená i beztoho, aby nastal pokrok v projektu. Alfy mají stavy, které lépe vyhodnotí pokrok na vývoji z různých pohledů a jako celek také poskytují informaci o celkovém stavu napříč všemi alfami. Alfy je také možné použít na:

- určení aktuálního stavu týmu, plánováni dalšího stavu a sledovaní postupu,
- přizpůsobení způsobů práce,
- určení zařádění jednotlivých členů týmu a povinnosti týmu.

SoC tak odděluje detailní reprezentaci (různé dokumenty) od alf.

Za pomoci jádra a SoC je také možné oddělit základní (to nejdůležitější) od detailního. Karty používané v Essence tak slouží, jako pomůcky na určení toho na čem se aktuálně pracuje a vedoucí týmu (projektu) také může za pomoci nich vést diskuse a případně poskytnut pomoc dle aktuálního stavu týmu. Karty tak obsahují to nejzákladnější, co tým potřebuje, detailnější informace je pak možné získat v jiných dokumentech (které jsou součástí projektu) nebo v knihách a pracích.

7.3 Klíčové novinky

SEMAT přistupuje k jádru a metodám softwarového vývoje s pomocí několika klíčových inovací:

- Být agilní při práci s metodami.

- Používat oddělení zájmů (separation of concerns), protože je to klíčový princip, který má dopad na celou iniciativu SEMAT.
- Definice separátních praktik nad jádrem umožňuje lehký vývoj a přizpůsobení metod dle specifických potřeb.
- Nalezením základních prvků softwarového vývoje a jejich zařazením do jádra vzniká základní kámen pro budování znalostí pro lepší, rychlejší vývoj a spokojenějšího zákazníka.

Být agilním při práci s metodami je dle SEMATu klíčové. To směruje k praktickým nástrojům, které pomáhají softwarovým specialistům v každodenní práci. Jsou to:

- Alfy: jsou základem pro pochopení různých dimenzí softwarového vývoje a výzev, které musí tým zvládnout.
- Alfa stavy: pomáhají profesionálům k pochopení průběhu a zdraví projektu.
- Karty (stavy a definice): dělají jádro hmatatelné (fyzické).
- Přesné praktiky: umožňují týmu se zaměřit na specifické detaily i mimo to co jim jádro nabízí.
- Možnost škálování: jádro a praktiky je možné používat v různě velkých týmech a také v různých projektech nebo organizacích, protože je lehce upravitelné.

V konečném důsledku je to zejména o lidech, jak spolu komunikují a zda chtějí pracovat efektivněji.

8 Závěr

Cílem práce bylo popsat:

- a stručně charakterizovat Essence, z čeho vznikl a k čemu slouží
- jak se s Essence a jeho jádrem pracuje
- jak je možné Essence využít v práci v týmu nebo při vývoji softwaru
- a určit, co nové tento standard přináší.

Cíle byly splněny a to následovně:

- V kapitole 2 byl stručně charakterizovaný Essence a bylo popsáno z čeho vychází a kdo je jeho autorem. Dále bylo v kapitole popsáno, z čeho se Essence skládá, a jak se s ním pracuje. Na to navázala další kapitola.
- V kapitole 3 bylo popsáno, jak je možné jádro využit v iteraci, jak tým za pomoci jádra plánuje, provádí kontrolu a přizpůsobuje svou práci.
- Kapitola 4 se věnovala procesu vývoje systému, která ukázala, jak je možné využít alfy a jejich stavy k práci na projektu.
- Kapitola 5 ukázala škálovatelnost jádra.
- Kapitoly 6 a 7 se pak věnovaly novým přínosům Essence, resp., jak se za pomoci Essence dívat na stávající metodiky pro vývoj softwaru.

Essence, jako již bylo napsáno v úvodě, je metodika na metodiku, která se snaží dát vývojářům a softwarovým profesionálům nástroj na tvorbu a správu metod na práci při vývoji softwaru. Dle našeho názoru, je to však opět metodika jenom pro metodiky a vývojářům by jenom komplikovala už tak dost komplikovanou práci (Essence udává příliš mnoho činností k tomu, abychom zjistili určitý stav nebo naplánovali nějakou část projektu).

Tahle práce je zaměřena více teoreticky a při dalším zkoumání problematiky Essence by bylo možné ji rozšířit o příklady z praxe, kde byla metodika nasazena, jak dané firmě pomohla apod.

9 Bibliografie

Jacobson, Ivar a Spence, Ian. 2014. YouTube. *Dr. Ivan Jacobson - The Essence of Software Enginnering: the SEMAT Approach.* [Online] Google TechTalks, 30. 07 2014. [Citace: 13. 11 2014.] https://www.youtube.com/watch?v=WNlERrVxYjs.

Jacobson, Ivar, a další. 2013. *The Essence of software engineering: applying the SEMAT kernel.* místo neznámé : Addison-Wesley Professional, 2013. 978-0-321-88595-1.

Object Management Group. 2014. Essence 1.0. *Documents Associated with Essence.* [Online] 02. 11 2014. [Citace: 04. 12 2014.] http://www.omg.org/spec/Essence/1.0/PDF/.

10 Seznam obrázků