Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informatiky

Metodika vývoje mobilních aplikací

Diplomová práce

Autor: Bc. Miroslav Holec

Studijní obor: Aplikovaná informatika

Vedoucí práce: doc. Ing. Filip Malý, Ph.D.

Hradec Králové březen 2015

Prohlášení

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne Miroslav Holec

Poděkování

Děkuji doc. Ing Filipovi. Malému, Ph.D. za odbornou pomoc, rady a vedení mé diplomové práce.

Anotace

Diplomová práce analyzuje současné tradiční i agilní metodiky a identifikuje předpoklady pro vývoj úspěšných mobilních aplikací. Na základě těchto předpokladů dále práce identifikuje procesy a fáze při vývoji mobilních aplikací a sestavuje z nich metodiku založenou na neustálé validaci vycházející z prototypu aplikace. Výsledkem je samostatně fungující metodika pro vývoj mobilních aplikací včetně možnosti propojení s dalšími vybranými agilními metodikami.

Annotation

Title: Methodology for Mobile Application Development

Diploma thesis is analyzing common traditional and agile methodologies and identifies assumptions for successful mobile application development. Based on these assumptions are identified processes and phases to create methodology for mobile application development focusing on continuous validation using application prototype. The result is independently working methodology for mobile application development including possibility to merging with another agile methodologies.

Obsah

Úvod

Značné množství oblastí lidské činnosti předpokládá řád a organizaci jako východisko pro nastávající úspěch. Lidé plánují svůj život, dovolené, dny i hodiny a analogicky společnosti vytváří a optimalizují firemní procesy, činnosti a organizují denní práci svých zaměstnanců. Vývoj software je komplikovaná disciplína, kterou bychom mohli přirovnat k řadě jiných procesů lidské činnosti. Celá řada lidských činností, jejichž výstupem je produkt, vyžaduje organizaci, finanční plánování, analýzu rizik a předpokládá odbornou schopnost zúčastněných pracovníků včetně jejich efektivní spolupráce.

Vývoj informačních systémů bezesporu připouští svobodu ve způsobu, jakým bude dosaženo cílů, ať už v procesech vládne pořádek či chaos. Jaká má být cesta k cíli vzhledem k finančním prostředkům a časovým možnostem? Odpověď na tuto otázku hledají tradiční i moderní agilní metodiky, jejichž snahou je proces vývoje software maximálně zefektivnit. Současné metodiky však kladou natolik přísné požadavky, že vývojářské týmy je nejsou schopné v jejich podobě aplikovat.

Relativně mladá oblast, vývoj mobilních aplikací, si žádá vzhledem ke svým specifickým odlišnostem vznik uceleného postupu, metodiky, která umožní vznik funkčního produktu vzhledem k dostupným finančním prostředkům a času. Klíčovými otázkami pro vznik takové metodiky je vhodná volba vývojových fází, proces střídání fází a jejich přizpůsobení ke specifikám mobilních aplikací ale i otázky související s testováním a procesem průběžného schvalování produktu. Řešením tohoto problému je ucelená Metodika vývoje mobilních aplikací.

Cíl práce a metodika zpracování

Cílem práce Metodika vývoje mobilních aplikací je vytvoření metodiky, která usnadní vývoj mobilních aplikací a pokryje kompletní proces vývoje od prvotní myšlenky až po zveřejnění aplikace či její odevzdání klientovi. Práce si klade za cíl, aby byla metodika flexibilní a umožnila vývojářským týmům a společnostem případné úpravy nebo koexistenci s dalšími agilními metodikami. Efektivita metodiky by měla spočívat především v získání nejlepších zkušeností ze stávajících metodik a jejich následné aplikace na mobilní vývoj. Metodika je určena především dodavatelům software bez ohledu na to, zda jej jich klientem zákazník či koncový uživatel.

Z metodického hlediska práce nejprve provede analýzu stávajících metodik, používaných pro vývoj software. V rámci této analýzy bude probíhat hledání užitečných praktik a procesů, které mají potenciál dopomoci k tvorbě efektivní metodiky v oblasti vývoje mobilních aplikací. Práce identifikuje všechny typické fáze vývoje aplikace včetně požadavků na jejich kvalitu dle vybraného modelu kvality a navrhne zodpovědné role, jež se budou na procesu vývoje aplikace podílet. Fáze budou logicky sestaveny tak, aby jejich návaznost dopomohla k nepřerušenému a funkčnímu procesu tvorby mobilní aplikace. Práce v závěru zmíní schopnost samostatného fungování metodiky i její možnost existence spolu s dalšími rozšířenými metodikami.

1. Analýza současných metodik

Ačkoliv je dnes obecně přijímáno paradigma, že budoucnost patří agilním metodikám, také tradiční metodiky mají při vývoji software stále své místo. Tradiční metodiky byly v posledních letech vytlačovány agilními metodikami, které reagují na některé podstatné nevýhody klasického přístupu. Tyto nevýhody však nutno poznamenat získaly na své podstatě s rozvojem informačních technologií, trendem vzniku projektů nabízejících unikátní hodnotu ale také díky užšímu propojení zákazníka či produktového vlastníka s koncovým uživatelem.

Dnešní software je velmi dynamický a spíše než snahou o jeho rozvoj do šířky (hledisko funkcionality) je snahou budovat systém do hloubky. Vydavatelé aplikací se snaží získat konkurenční výhodu zpětnou vazbou od koncových uživatelů a spíše než dodávat funkce nové, dávají přednost vylepšování funkcí stávajících. Proces vývoje aplikací je procesem nekonečným, nové verze softwarových systémů jsou doručovány prakticky neustále a plánování vývoje probíhá permanentně v podobě iterací.

Vývoj mobilních aplikací tak, jak je v současné době známe, je velmi mladá oblast. I když první chytrá zařízení se na trhu objevila před více než 20 lety, skutečnou revoluci na trhu odstartovaly až operační systémy iOS a Android. Trendem dnešní doby jsou dle žebříčků oblíbených aplikací kvalitní jednoúčelové aplikace, jejichž specifikem je jednoduchost, minimum funkcí ale zároveň vysoká intuitivnost a ambient inteligence plynoucí z přílivu nových technologických novinek. (1)

1.1. Tradiční metodiky

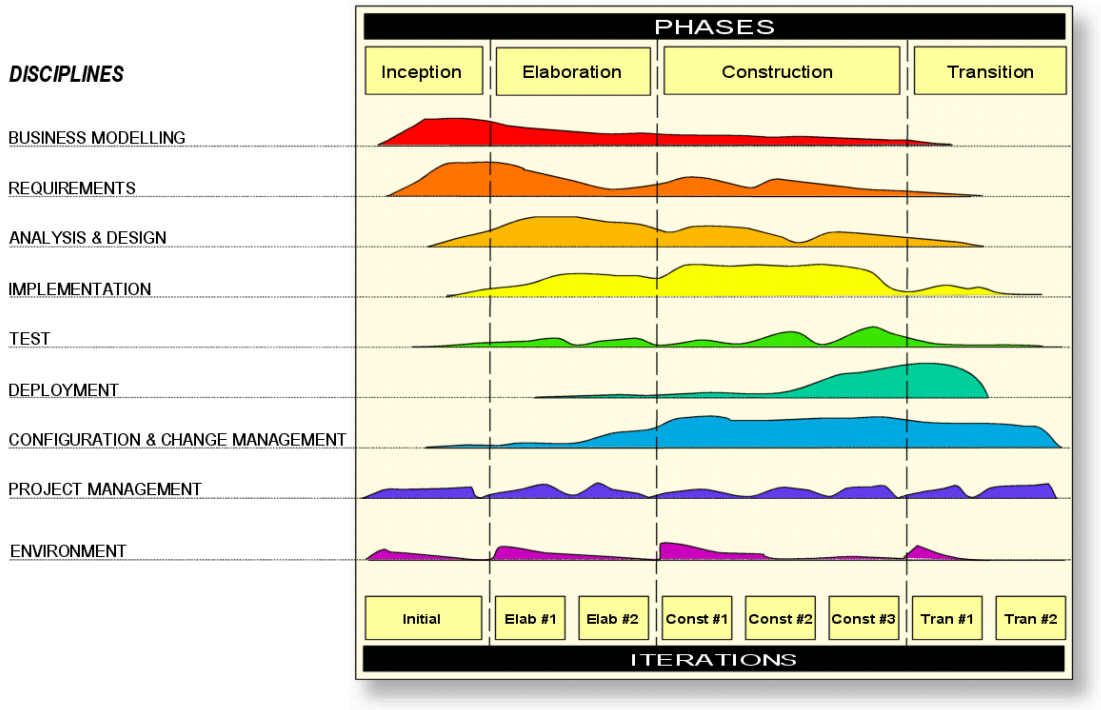
Mezi obecně nejpoužívanější tradiční metodiky vývoje software patří vodopádový a spirálový model. Specifickou kategorii tvoří metodiky RUP a alternativní UP, které z vodopádového modelu sice vycházejí ale jejich přístup je více iterativní. Odlišnost jednotlivých metodik spočívá v několika oblastech tvořících jejich koncept. Jedním z nejdůležitějších rozdílů je způsob, jakým na sebe jednotlivé fáze vývoje navazují. Vodopádový model je založen na myšlence střídání

fází od původní specifikace až po integraci a údržbu, nicméně již neřeší zpětnou vazbu a následné změny jako reakci na podněty zákazníka. Pokud dojde ke změně požadavků, spouští se znovu celý proces od analýzy požadavků až po dodání řešení. (2)

Zajímavější spirálový model každou fázi navíc rozděluje na plánování, stanovení cílů, analýzu rizik a následný vývoj. Právě analýzou rizik se také jednotlivé metodiky vývoje mohou významně lišit a spirálový model klade na tuto oblast velký důraz. Co však mají obě metodiky společné je jich izolovanost od zákazníka a dodání hotového produktu, bez toho aniž by zákazník mohl do procesu průběžně zasahovat. Pokud již k nějakému zásahu dojde, obvykle to má negativní vliv na dobu dodání a cenu řešení. Nejen, že se tak zvyšuje riziko vzniku produktu, který se rozchází s představou zákazníka, ale může dojít i k nepřijetí vyvíjeného řešení ze strany koncových uživatelů. (2)

1.2. RUP a UP

Metodika RUP byla vytvořená společností Rational Software Corporation, která se stala v roce 2003 součástí společnosti IBM. RUP reprezentuje unifikovaný proces vývoje aplikací a jeho snahou je vývoj software ve čtyřech po sobě iterujících krocích. Na pozadí těchto čtyř kroků je v čase kladen důraz na jednu ze šesti klíčových oblastí vývoje od modelování až po doručení řešení zákazníkovi. Typické pro RUP je vyjádření procesu diagramem dle Obr. 1.



Obr. 1: Realizace RUP, zdroj: ibm.com

Dále je metodika RUP postavena na velmi formální dokumentaci, která na jedné straně usnadňuje řízení projektu, na straně druhé však vyžaduje mnoho energie a času z celého procesu. Klíčový je pro RUP také proces testování, které probíhá neustále, přičemž nejintenzivnější je ve fázi předání řešení zákazníkovi. Společně s formální dokumentací tato metodika umožňuje vývoj aplikací, jejichž nedostatky bývají odstraněny již v průběhu implementace, kdy je jejich náprava relativně levná.

UP reprezentuje unifikovaný a generický proces vývoje software stejně jako v případě RUP ale s tím rozdílem, že RUP je komerční implementací s nepatrnými rozdíly. Metodika UP definuje iterace s pěti klíčovými aktivitami: požadavky, analýzy, návrh, implementace a testování. Typikem iterací je, že se mohou do určité míry i překrývat. Iterace pak na sebe navazují na pozadí čtyřech základních fází procesu vývoje: zahájení, rozpracování, konstrukce a zavedení. (4)

1.3. Agilní metodiky

Agilní metodiky představují odlišný přístup vůči všem tradičním. Zatímco tradiční metodiky definují funkcionality, kterým přizpůsobují všechen svůj čas a zdroje, agilní metodiky pracují výhradně s časem a přidělenými zdroji bez ohledu na to, jaké funkcionality vzniknou. Podstatný společný rys agilních přístupů je úzké propojení klienta či koncového uživatele s vývojářským týmem. Právě této úzké vazbě jsou přizpůsobeny fáze vývoje, střídající se obvykle v iteracích. Tyto iterace jsou v různých metodikách nazývány odlišně.

Pro pochopení agilních metodik je vhodné se pozastavit u samotného manifestu agilního vývoje software, jehož autory jsou přední světoví softwaroví inženýři. Všechny metodiky jsou postaveny na těchto pravidlech s tím, že nejblíže jim má tzv. extrémní programování, jinak také XP.

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

Jednotlivci a interakce před procesy a nástroji

Fungující software před vyčerpávající dokumentací

Spolupráce se zákazníkem před vyjednáváním o smlouvě

Reagování na změny před dodržováním plánu

Jakkoliv jsou body napravo hodnotné, bodů nalevo si ceníme více.“ (3)

1.3.1. Extrémní programování

Metodika vytvořená Kentem Beckem, jedním ze zakladatelů Agile Manifesto, pracuje se zdrojovým kódem aplikace, jako jedním z hlavních zdrojů informací. Zdrojový kód je ve společném vlastnictví všech vývojářů a reflektuje potřeby uživatelů, které jsou vždy definované jako uživatelské příběhy (User Stories). Každý uživatelský příběh popisuje potřebu zákazníka. Z hlediska UML se jedná o zjednodušený a neformální scénář typové úlohy, jenž má formu popisu o délce tří vět až jednoho odstavce.

Vývojářský tým v pravidelných iteracích o délce jednoho až čtyř týdnů volí ve spolupráci se zákazníkem uživatelské příběhy k vyřešení a jako východisko k jejich naplnění považuje jednotkové testy (Unit Testy). Jednotkové testy pokrývají kompletní funkcionalitu aplikace, případně pomáhají řešit vzniklé chyby. Velmi typický je pro XP i přístup k délce iterace. Zatímco v počátku vývoje jsou iterace delší, v průběhu zprovozňování se délka iterací zmenšuje a až trojnásobně se zvyšuje frekvence vydávání nových verzí software.

Vývoj software je založen na párovém programování, jehož cílem je zvýšení kvality kódu a rozvíjení zkušeností vývojářů. Dva vývojáři se pravidelně střídají v pozici aktivního vývojáře a partnera (řidič a navigátor), který má za úkol vznikající kód kontrolovat nebo konzultovat. Díky této metodice je tým schopen bezesporu rychleji porozumět práci svých kolegů, nalézat efektivnější řešení a snížit riziko vzniku chyby. To má pozitivní důsledky z finančního hlediska, protože odhalení a náprava chyby v produkčním prostředí může stát o několik řádů více finančních prostředků než zachycení stejné chyby v průběhu implementace.

Metodika předpokládá velmi úzkou spolupráci zákazníka s vývojářským týmem, označovanou jako „zákazník na pracovišti“. Společně s předpokladem dvou vývojářů u jednoho stroje tak klade podmínky, které velké množství týmů není schopné naplnit. Kent Beck sám zmiňuje, že XP je v detailech jednoduché ale obtížné při vykonávání. V případě menších projektů a vývojářských týmů umožňuje rapidně rychlý vývoj a díky párovému programování a společnému vlastnictví kódu usnadňuje předávání znalostí mezi vývojáři. (4) Předpokladem je však důvěra mezi produktovým vlastníkem a vývojářským týmem stejně jako v dalších agilních metodikách.

1.3.2. TDD

Velmi podobné myšlenky z hlediska procesu vývoje předkládá vedle XP i metodika TDD, jež vychází z myšlenky, že testy musí vzniknout před samotným funkčním kódem. Testy jsou první fází každé iterace a ve své podstatě definují podobu systému, respektive určité funkcionality. Proti testům je následně napsán

funkční aplikační kód, který je ve třetím kroku zrefaktorovaný, aby byl co nejjednodušší a nejefektivnější. Vzhledem k tomu, že právě testy jsou klíčovou oblastí v případě zajišťování kvality a správné funkcionality, metodika TDD se právě od testů odvíjí.

Kromě testů jednotkových metodika pracuje s testy akceptačními a následně i integračními. Principy TDD ústí v myšlenku, že vznikající řešení je po dokončení fází implementace a návrhu bezchybné a otestované. Vzhledem k tomu, že popis chování systému je tvořen testy, TDD již dále nevyžaduje další dokumentaci nebo formální specifikace systému.

Spekulativní může být otázka, nakolik mohou samotné testy ovlivnit kvalitu a bezchybnost aplikací, vytvářených metodikami XP a TDD. Z logiky věci totiž chybně definované testy znamenají i chybně definované chování systému a připouštějí nežádoucí připuštění testu. Také správně myšlený ale chybně nebo nedostatečně implementovaný jednotkový test vytváří prostor pro vznik chyb.

1.3.3. SCRUM

SCRUM je agilní metodika obsahující mimo jiné sadu doporučení, na základě kterých by měl fungovat tým jako celek a dále popis odpovědností všech zúčastněných osob. Typickou rolí ve SCRUMu je vlastník produktu (Product Owner), jehož úkolem je definovat vizi produktu, určovat priority a získávat zkušenosti a zpětnou vazbu od koncových uživatelů a zákazníků. Velmi specifickou rolí je také Scrum Master, jehož úkolem je zajistit správné fungování týmu, organizovat pravidelné schůzky, motivovat tým k lepším výsledkům a pomáhat se týmu soustředit na zvolené cíle.

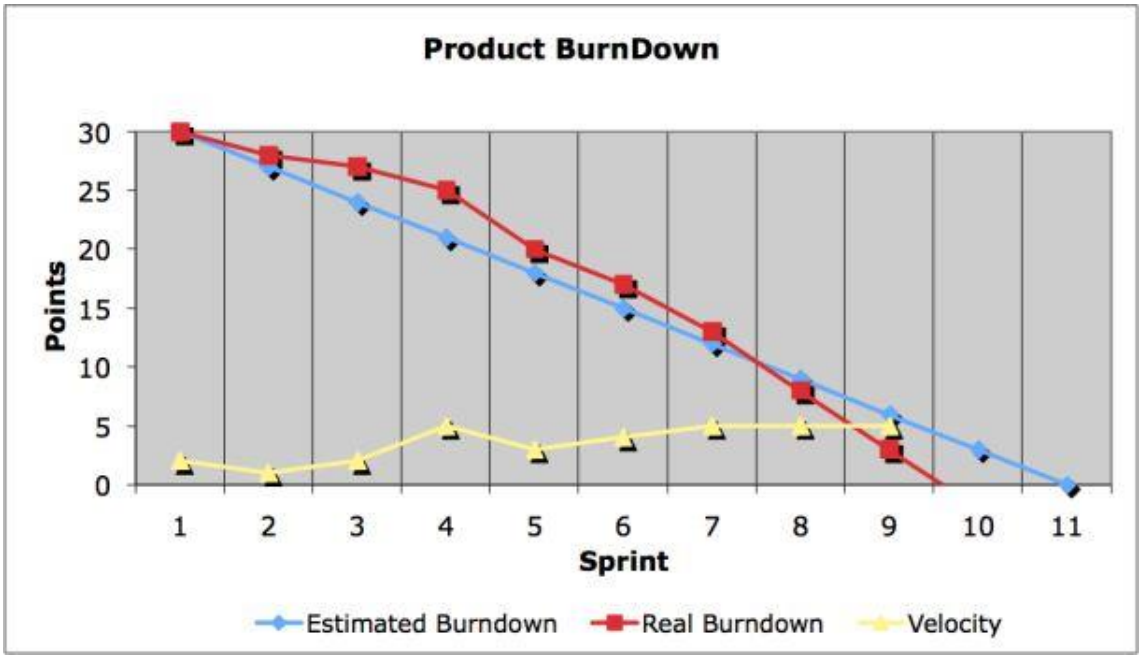
Ve SCRUMu se všechny požadavky od Product Ownera zapisují do produktového Backlogu, z něhož Product Owner vybírá položky dle priority a rozhoduje tak o tom, jaké funkcionality budou vyvíjeny v rámci tzv. Sprintu. Sprinty trvají stejně jako iterace v XP mezi 1 až 4 týdny. Komunikace uvnitř týmu je zajištěna dlouhými strategickými poradami na začátku nových Sprintů, retrospektivními poradami a krátkými pravidelnými denními poradami (Stand-up

meeting), které pomáhají odstranit největší překážky při vývoji aktuálních funkcionalit.

SCRUM se snaží do určité míry propojit „hravost s užitečností“. Například tzv. Planning poker umožňuje specifickým způsobem odhadovat náročnost implementace různých částí systému. Na základě do poslední chvíle neznámých časových předpokladů jednotlivých členů týmu, proběhne první odhad, který je následně zpřesňován při konverzaci nad uvažovaným problémem. V konečné fázi tým jednohlasně odhadne náročnost vyvíjené funkce.

Stejně jako v případě XP je nutné úzce pracovat se zákazníkem, který by se měl prakticky denně účastnit meetingů a ovlivňovat následný sprint. Dalším faktem je, že SCRUM není jednoduše realizovatelný a vyžaduje mnoho času a disciplínu, kterou se obvykle vyplatí vynaložit na střední a velké projekty. V neposlední řadě pak tato metodika vkládá vysokou důvěru v členy týmu a výrazně abstrahuje od formalizace požadavků a vzniku dokumentace. Bez přítomnosti zákazníka nebo v případě změny osoby v roli Product Ownera se vývoj řešení může stát neudržitelný a nepřehledný.

Jako jiné agilní metodiky i SCRUM zakazuje práci přesčas, jelikož nedostatečně odpočatý zaměstnanec odvádí méně práce nebo práci při nižší kvalitě či s chybami. Pokud není v silách vývojářského týmu stihnout všechny úkoly ve Sprintu nebo naopak tým pracuje nad očekávání rychleji, může Scrum Master úkoly odebírat, resp. přidávat. Ke sledování postupu v průběhu Sprintu slouží Burndown chart. (6)



Obr. 2: BurnDown diagram, zdroj: scrum-institute.org

1.3.4. Lean

Metodika Lean je velmi podobná doposud zmíněným agilním metodikám a odlišnosti lze najít především jen v názvosloví a filosofii. Lean je postavena na myšlence odstranění zbytečných funkcionalit, které navyšují náklady na provoz řešení, aniž by přinesly přidanou hodnotu. V praxi se Lean snaží nastavit procesy tak, aby byly zavčas odhaleny chybné myšlenky a neperspektivní směr vývoje a energie mohla být vynaložena do vývoje prospěšnějších funkcí.

Filosofie metodiky Lean vyžaduje velmi úzké zaměření na vyvíjené funkce a často preferuje tyto vyvíjené funkce minimalizovat. Doporučuje stanovit hypotézy, za kterých bude funkcionalita nebo myšlenka prohlášena za validní a teprve v momentě ověření hypotézy se obvykle provádí další rozvoj funkcí. (8)

1.3.5. FDD

Vývoj řízený vlastnostmi, jinak také „Feature-driven development“ je agilní metodika hlouběji zaměřená na proces návrhu a samotné implementace. Ačkoliv metodika pokrývá jen některé životní části vzniku software, neustále se snaží tyto klíčové oblasti kontrolovat. Vývoj software probíhá na základě přírůstků v podobě

různých funkcí (Features), které mají pro zákazníka nebo konečného uživatele užitnou hodnotu. Každá vlastnost je nejprve v rámci fáze návrhu modelována a následně probíhá její implementace.

V metodice FDD je také definována celá řada rolí a jejich zodpovědností. Dohled nad projektem má projektový manažer, proces vývoje řídí manažer vývoje. Jemu je dále zodpovědný hlavní vývojář nebo více vývojářů, kteří řídí ostatní programátory. Programátoři jsou zároveň vlastníci tříd a jejich zodpovědností je rozvoj své třídy, v širším slova smyslu funkcionality nebo vlastnosti systému. FDD se v tomto ohledu liší od metodiky XP, ve které je zdrojový kód ve společném vlastnictví týmu a odpovědnost za konkrétní funkci není pouze na samotném vývojáři. (8)

Metodika definuje i další vedlejší role, nicméně otázkou zůstává, zda právě pro mobilní vývoj není vhodnější od velkého množství rolí a hierarchické struktury upustit k menšímu množství obecnějších a klíčových rolí postavených na stejnou úroveň. Myšlenka modelování ve fázi návrhu není v případě mobilních aplikací vždy ideální, jelikož způsob realizace vybrané funkce bývá známý až těsné před implementací. Modelování by v takovém případě muselo zahrnovat spolupráci všech rolí zodpovědných za všechny fáze až do momentu technické realizace.

1.4. Zhodnocení metodik

Na základě popisu agilních i tradičních metodik vývoje software lze nyní provést zhodnocení klíčových oblastí, ve kterých se metodiky liší.

1.4.1. Proces vývoje

Tradiční i agilní metodiky se dramaticky rozcházejí především v oblasti realizace procesu vývoje. V případě tradičních metodik můžeme pozorovat, že vývoj probíhá na základě na sebe navazujících částí, které očekávají kompletní dokončení předešlé fáze. Z principu to může znamenat, že chyba v dokumentaci při realizaci fáze implementace řešení může mít za následek nutné přepracování všech navazujících fází. Zásadní problém může nastat v případě větších systémů, protože

každá fáze je časově velmi náročná a finančně nákladnější, oproti malým iteracím agilních metodik.

V případě agilního vývoje je software dělen na „stavební bloky“, které jsou vytvářeny v neustálých iteracích a tudíž tento problém obvykle postihne jen jeden vyvíjený blok. Agilní metodiky se liší především názvoslovím. Stavební bloky jsou nazývány funkcemi, uživatelskými scénáři nebo uživatelskými příběhy a proces střídání fází vývoje je obvykle označován jako iterace nebo sprint.

Typické je pro agilní metodiky i neustálé doručování software buď formou Continuous Delivery nebo Continuous Deployment. Mezi výhody těchto přístupů jmenujme redukci množství chyb, méně stresu v souvislosti s přípravou nových produkčních verzí software nebo větší flexibilitu vydávání nových verzí. Continuous Deployment nakonec preferuje automatizaci celého procesu pomocí různých nástrojů tak, aby každá změna v software prošla procesem ústícím v novou produkční verzi. (5)

1.4.2. Lidské zdroje

Významně se jednotlivé metodiky liší také rolemi, kterých nabývají různé zúčastněné osoby při vývoji software. Tradiční metodiky uznávají spíše hierarchickou strukturu spolupráce, zatímco techniky agilní doporučuji stavět role na stejnou úroveň a vyznačují se také svou specifičností (např.: zmíněný Scrum Master). Kromě rolí ale nalezneme odlišnosti i v komunikaci. Agilní metodiky dávají přednost přímé komunikaci se zákazníkem před tvorbou dokumentace, která může být pro zákazníka hůře pochopitelná. Uveďme i specifickou vlastnost extrémního programování – párový vývoj. (5)

1.4.3. Dokumentace

Agile Manifesto dává přednost spolupráci se zákazníkem před tvorbou dokumentace a tento bod se promítá prakticky do všech metodik. Tradiční metodiky striktně dodržují formální tvorbu dokumentace, metodiky RUP a UP střízlivě využívají UML pro popis aplikačního software a agilní metodiky víceméně dávají

přednost méně formální dokumentaci nebo dokumentaci žádné. Základním prvkem jsou zmíněné uživatelské příběhy, které mají obvykle podobu krátkého slovního popisu. Všem agilním metodikám však subjektivně chybí určitá retrospektiva v oblasti dokumentace, která by popisovala architekturu vyvíjeného řešení a komplikované komponenty systémů nebo komponenty komunikující s externími systémy. Dává nicméně smysl, aby vznikl prostředek, který blíže specifikuje celkovou základní představu produktového vlastníka o vyvíjeném řešení.

1.4.4. Měření a analytika

Některé metodiky, především Lean, jsou orientované na vývoj inovací a snaží se průběžně hodnotit přínos vytvářených funkcí v souvislosti s náklady a hodnotou pro koncového uživatele. Tento přístup může mít významný přínos pro technologické startupy a při vývoji software „dovnitř“ – tedy pro firmu samotnou.

V případě vývoje řešení pro zákazníka, který dané řešení distribuuje konečnému uživateli, se tento metodický postup může projevit negativně. Vyžaduje totiž zpětnou vazbu, která nesmí být ovlivněna vývojem jiných funkcí a v důsledku toho je nutná úzká komunikace a flexibilita zákazníka. Nesmíme také zapomenout na fakt, že u rozsáhlých informačních systémů může být měření přínosu jednotlivých funkcí komplikované, drahé v souvislosti s nutností kvantifikace a ve vzácných případech i nerealizovatelné. Přes všechny zmíněné překážky může být měření užitečnosti funkcí užitečné rozšíření procesu vývoje aplikací.

1.4.5. Testování

Poslední, avšak neméně důležitou oblastí je samotné testování software. Uvažujme například jednotkové testy. Agilní metodiky XP a TDD doporučují vyvíjet aplikační software metodou „Test-First“, tedy nejprve psaním testů a následně psaním funkcionalit, které odpovídají definici testů. Jiné metodiky ale mají natolik odlišnou filosofii, že v oblasti testování žádná konkrétní doporučení nemají (SCRUM). Jiným typům testů v souvislosti s fyzickým vývojem software se metodiky většinou nevěnují, ale v mnohých případech se můžeme setkat s procesem validace řešení s představami koncového uživatele nebo zákazníka. Subjektivně testování by

mělo zastávat v každé metodice důležitou roli a mělo by testovat všechny klíčové okamžiky v procesu výroby.

1.5. Problém metodik

Agilní metodiky se v detailech liší, přestože jejich myšlenka je zcela totožná. Extrémní programování se zaměřuje na kvalitu zdrojového kódu a společně s TDD preferují Test-First přístup na úkor dokumentace řešení. SCRUM je hravá metodika avšak stejně jako XP vyžaduje úzkou spolupráci zákazníka a především nepokrývá celý proces vývoje software. Metodiky s orientací na plánování neřeší proces z hlediska technologických fází výroby (např.: UX) ale spíše jen z abstraktního hlediska, aby byla zachována jejich univerzálnost. Sporná je i otázka testování, které je často pojato jako izolovaná činnost na konci iterace namísto jeho začlenění do všech fází ve formě konkrétních akceptačních kritérií. V důsledku výše uvedeného má celá řada vývojářských týmů problém zvolit pouze jednu metodiku a tu striktně dodržovat.

1.6. Závěrečné shrnutí

Agilní metodiky při vývoji software přináší dle některých zdrojů až trojnásobně větší úspěch, než metody založené na vodopádovém vývoji. (3) Výhodou a přínosem agilního přístupu je bezesporu užší propojení se zákazníkem a s užitím iterativního vývoje i možnost flexibilních změn produktu již v raných fázích vývoje. Tradiční přístupy představují řešení, které je značně flexibilní a snadno implementovatelné, zatímco současné agilní metodiky kladou na vývojářské týmy těžko překonatelná kritéria. Výhodou agilních metodik je však schopnost předejít vývoji zbytečných funkcí právě díky úzké spolupráci se zákazníkem.

Příchod agilních metodik přinesl větší úspěšnost projektů jako takových, avšak zavedl nový trend v podobě neschopnosti týmů agilní metodiku dodržovat. Vývojářské týmy tak sice reprezentují skupinu agilních vývojářů a dokážou tvořit konkurenceschopné produkty, ale ve skutečnosti selhávají v oblasti disciplíny

a dodržení pravidel vybraných metodik. Agilní metodiky připouštějí značné množství svobody, ale často působí z pohledu zákazníků „neformálně“. (7)

Není pochyb o tom, že každý tým si najde svou metodiku, která mu bude více či méně vyhovovat. Z hlediska vývoje mobilních aplikací však všechny metodiky přinášejí především filosofický směr než specializovaný technický postup, který by vedl k vytvoření aplikace dle vybraných kvalitativních kritérií. Metodiky více technicky zaměřené vnáší doporučení především v oblasti programování, ale již se nezmiňují o vizuální stránce software nebo například uživatelské přívětivosti. Bezesporu lze však konstatovat, že jakákoliv metodika odvozená z principů Agile Manifesto má potenciál zvýšit pravděpodobnost úspěchu vyvíjeného řešení.

2. Mobilní aplikace

Američtí uživatelé stráví používáním mobilních zařízení každý den bezmála jednu hodinu. (10) Trhy s mobilními aplikacemi se neustále rozvíjí a nabízejí nové užitečné aplikace i hry. V současné době patří mezi tři nejvýznamnější trhy App Store od společnosti Apple, Google Play a Store od Microsoft Corporation. Na těchto virtuálních tržištích jsou k dispozici statisíce aplikací a mobilních her, jejichž způsob vzniku a technologické zázemí se významně liší. Tato práce se soustředí pouze na hledání metodiky pro oblast nativních mobilních aplikací. V této části jsou popsány současné trendy v oblasti vývoje mobilních aplikací a dále jsou identifikovány předpoklady tzv. „úspěšné aplikace“. Práce v dalších kapitolách již vychází z těchto předpokladů a požadavků na vlastnosti mobilních aplikací.

2.1. Nativní webové aplikace

Pokud se vývojářský tým rozhodne vytvořit novou mobilní aplikaci, má ve své podstatě tři možnosti. Tou nejjednodušší je vytvoření tzv. responsivní webové stránky. Responsivní webové stránky nejsou aplikace v pravém slova smyslu, jsou zcela nezávislé na používaném zařízení a de facto jsou pouze webovou stránkou, která je uzpůsobena ploše displeje s pomocí CSS stylů. Tyto responsivní webové stránky využívají standardně technologii HTML 5, jsou zobrazovány v mobilním webovém prohlížeči, ale dostupné jsou i skrze prohlížeče na desktopu. V případě systémů iOS a Windows Phone je používáno vždy jen jedno vykreslovací jádro, ať už je použit jakýkoliv prohlížeč webu. Tím se pro tyto dva operační systémy proces optimalizace značně zjednodušuje. Nevýhodou tohoto řešení je, že uživatel musí navštívit za účelem každého použití domovskou URL adresu aplikace.

Druhá, sofistikovanější varianta je tvorba mobilní aplikace, která obsahuje sama o sobě vlastní prohlížeč. Taková mobilní aplikace může být do uživatelského zařízení stažena z virtuálního tržiště a může využívat celou řadu hardwarových funkcí zařízení. Výhodou tohoto přístupu je společný vývoj pro všechny platformy, obvykle s pomocí vybraného frameworku a IDE. Nevýhodou těchto aplikací je stejně jako v předešlém případě komplikované přizpůsobení vzhledu grafickým

standardům platformy a stále chybějící možnost aplikací spolupracovat úzce s operačním systémem zařízení.

Přestože HTML 5 již umožňuje vytvářet webové aplikace, které jsou schopné provozu i bez konektivity, nativní aplikace stále nabízejí v této oblasti více možností. Jejich další nesporné výhody jsou podstatně větší a bezpečnější úložiště dat uvnitř mobilního zařízení a vyšší výkonnost. Chce-li vývojářský tým vyvíjet mobilní aplikace pro libovolnou platformu, má obvykle i řadu nástrojů třetích stran, které vývoj usnadňují. Výhodou nativních aplikací je také užší propojení s tržištěm a zabudování tzv. In-App Purchases, tedy funkcí, které se dají do aplikace dokupovat. Nativní aplikace v neposlední řadě umožňují i snadnou implementaci různých reklamních systémů. Společným rozdílem všech mobilních aplikací oproti desktopovým je typický způsob jejich ovládání pomocí gest prsty nebo s pomocí dalšího hardware (zvuk) a senzorů.

2.1.1. Aktualizace

Nativní webové aplikace jsou instalovány přímo do zařízení uživatele a jedná se z podstaty o plnohodnotné aplikace typu tlustý klient. To nese oproti webovým aplikacím řadu výhod i nevýhod. Jednou z významných nevýhod je proces aktualizace. Zatímco v případě tenkého klienta stačí aktualizovat aplikaci na sdíleném (webovém) serveru a tuto verzi mají okamžitě všichni uživatelé k dispozici, proces aktualizace nativních aplikací je komplikovanější. Vydavatel musí splnit řadu požadavků proto, aby byla jeho aplikace nebo aktualizace přijata na virtuální tržiště, dále musí čekat na schválení správcem a poté na doručení notifikace uživatelům, kteří aplikaci využívají. V neposlední řadě se aplikace dostane ke koncovému uživateli až v momentě, kdy si ji skutečně uloží do svého zařízení a povolí aktualizaci. Je-li společně s mobilní aplikací vyvíjeno i aplikační rozhraní, je pak nezbytné, aby z výše jmenovaného důvodu byla zachována zpětná kompatibilita. Výhodou aplikací typu tlustý klient je především plnohodnotná schopnost práce off-line.

2.2. Vzhled a funkcionalita

Každá mobilní aplikace, s níž uživatel disponuje je tvořená dvěma základními oblastmi. Tou první, kterou uživatel obvykle nejvíce vnímá je vzhled samotné aplikace, případně rozložení prvků, ovladatelnost a intuitivnost. (14) Pro uživatele je klíčové, aby aplikaci rychle pochopil, byl schopen ji podvědomě používat a aby aplikace realizovala jeho potřeby. Na straně druhé jsou technické funkce a samotné technologické zázemí, které uživatele do větší míry nezajímá nebo jej vnímá prostřednictvím vizuálních funkcí a chování. Základní požadavek na technickou část aplikace je, aby správně fungovala a pokud možno nezpůsobovala neočekávané výjimky. Také do této oblasti spadá využití konkrétního hardware a jeho prostředků nebo senzorů. Využití senzorů nebo dalších externích technologií za účelem zvýšení intuitivního chování aplikace spadá do oblasti ambient intelligence. Technologická část ovlivňuje v neposlední řadě rychlost aplikace nebo například data a jejich strukturu či samotnou dostupnost celého řešení. Z pohledu dodavatele je technické řešení důležité z důvodu dalšího efektivního rozšiřování aplikace.

2.3. Mobilní platformy

Jednotlivé mobilní platformy prošly za poslední roky dlouhým technickým vývojem, avšak z filosofické podstaty se od sebe liší od samého počátku.

2.3.1. Android

Operační systém Android byl společností Google ohlášen 5. listopadu 1997. Jeho určení je pro mobilní zařízení a typická je pro něj otevřenost pro vývojáře i průmysl obecně. Android můžeme nalézt v mobilních telefonech, v tabletech nebo v pozměněné verzi i v chytrých televizorech a hodinkách. Na Google Play bylo k červenci 2014 více než 1,3 milionu dostupných aplikací. Otevřenost Androidu vítá velká část uživatelů, kteří prostřednictvím utilit dostávají možnost si systém přizpůsobit ke svému obrazu, vývojáři vyvíjející utility pro změnu chování operačního systému nebo samotní distributoři, kteří vlastnosti systému upravují dle svých marketingových potřeb. Daní je nicméně riziko uvedení operačního systému

do nekonzistentního stavu právě používáním neověřeného software nebo ve druhém případě závislost uživatele na aktualizacích od dodavatele pozměněného systému (nejčastěji mobilním operátorovi). (7) (8)

Platforma Android za dobu svého působení našla útočiště u mnoha výrobců prodávajících zařízení ve všech cenových kategoriích od low-end mobilních telefonů až pro velmi výkonná zařízení. Tato fragmentace trhu zároveň přinesla vývojářům nemalé komplikace. Verze Android jsou vydávány relativně často a společně s velkým množství zařízení vzniká kombinace mnoha konfigurací, které je v rámci vývoje nutné důkladně otestovat a najít pro ně technologický kompromis.

Z vizuálního hlediska prošel systém dlouhým vývojem, během kterého se řada grafických doporučení v rámci verzí průběžně měnila. Společně s benevolencí tržiště tak existují statisíce aplikací postrádajících nebo ignorujících grafická doporučení platformy. Nejkvalitnější aplikace v tržišti se nicméně drží jednotného vzhledu i typografie reprezentované společností Google na oficiálních stránkách pro vývojáře a designéry.

2.3.2. iOS

Společně s prvním velmi populárním zařízením iPhone 1. generace uvedl Apple i verzi operačního systému iOS 3. Zatímco Google se svým systémem Android dává uživatelům volnou ruku, společnost Apple věnuje značné úsilí k získávání zpětné vazby a dodání hotového operačního systému, který má být natolik intuitivní, že by uživatelé neměli cítit potřebu cokoliv změnit. Systém iOS je dostupný nejen na mobilních telefonech, ale i na tabletech iPad a přehrávačích iPod. Jednotlivé verze operačního systému jsou obvykle distribuovány vždy společně s novou verzí mobilního zařízení s tím, že starší generace zařízení mohou provést update na novou verzi OS. Pro systém iOS vznikají neustále i neoficiální utility pozměňující chování systému, označované jako Jailbreak. Tyto utility umožňují náročnějším uživatelům přístup k rozšířeným nastavením nebo možnost instalace aplikací mimo tržiště App Store. To vše za cenu nestability systému a závislosti na aktualizacích konkrétního vydavatele Jailbreaku. (7)

Skutečnost, že společnost Apple vydává pouze omezený počet zařízení a je zároveň i výrobcem operačního systému má za následek potlačení fragmentace a výrazně usnadňuje vývojářům testování, jelikož pro regulérní test stačí pouze jediné zařízení se systémem iOS. Pro tržiště App Store platí přísné podmínky pro schválení aplikace a dá se s nadsázkou říci, že toto tržiště neobsahuje mnoho nekvalitních aplikací vzniklých jen v rámci testování možností platformy nezávislými vývojáři. Oproti aplikacím na platformě Google vyžaduje Apple, aby aplikace obsahovaly relativně konzistentní ovládání a design dle doporučených pokynů. Přestože je trh s iOS aplikacemi menší, uživatelé používáním iOS zařízení tráví denně o 35% více času než uživatelé systému Android. App Store dosáhl v červenci 2014 hranice 1,2 milionu aplikací. (8) (9) (10)

Vzhled aplikací na platformě iOS je řízen na základě tzv. iOS Human Interface Guidelines. Ty stejně jako v případě systému Android definují vzhled mobilních aplikací včetně prvků, typografie nebo chování při různých dotykových událostech.

2.3.3. Windows Phone

Operační systém Windows Phone je nástupcem systému Windows Mobile. Zatímco Windows Mobile mohl působit jako odlehčená a mobilním zařízení přizpůsobená verze běžného operačního systému Microsoft Windows, nová verze Windows Phone vydaná v roce 2010 nabídla zcela odlišné rozhraní. Windows Phone je v současné době stále velmi mladá mobilní platforma s nejmenším virtuálním tržištěm o více než 300 tisících aplikací k červenci 2014. Společně s iOS sdílí filosofii, v rámci kterých uživateli doručuje hotový operační systém a znemožňuje jakýmkoliv způsobem měnit vzhled a chování systému (mimo uživatelská nastavení). (9)

Vzhled aplikací je od samého uvedení první verze tohoto systému postaven na tzv. grafickém jazyce Metro založeném především na typografii. Dle tohoto jazyka je stěžejním grafickým prvkem obsah odekorovaný čitelnou typografií. Jazyk je v současné době rozšířen i na operační systém Windows a online aplikace z portfolia

společnosti Microsoft. Grafické rozhraní Metro od samého vydání vyvolává spíše rozporuplné reakce kritiků a jeho rozšíření nepomohlo ani přejmenování na Modern UI, které se uskutečnilo v polovině propagační kampaně.

Přestože Windows Phone zpočátku netrápil problém fragmentace zařízení jako v případě systému Android, v současné době jsou na trhu již tři rozdílné verze operačního systému distribuované na low-end i high-end zařízení. Díky tomu mohou náklady na vývoj pro tuto platformu vzrůst v procesu testování aplikací a jejich přizpůsobování vzhledem k různým verzím operačních systémů a zařízení.

2.4. Úspěšné moderní aplikace

Má-li být metodika přizpůsobena vývoji úspěšných aplikací, je vhodné se pozastavit u úspěšných aplikací na virtuálních tržištích a odvodit jejich společné rysy, které mohou ovlivnit proces výroby aplikace. Mezi úspěšné aplikace zařaďme nejoblíbenější aplikace na virtuálních tržištích k únoru 2015 v rámci českého trhu a analyzujme je. Mezi tyto aplikace lze skrze platformy zařadit: Messenger, Facebook, WhatsApp, Viber, Instagram, Shazam, Sazka, Adobe Reader. Ačkoliv za úspěchem těchto aplikací stojí silná komunita, bezesporu lze najít ke každé z těchto aplikací i řadu méně úspěšných alternativ. V čem se tedy nejúspěšnější aplikace liší a v čem spočívá jejich úspěch?

Bezesporu je pro každého uživatele důležitá ovladatelnost a intuitivnost aplikace. Především aplikace, které se používají „v akci“, by měly být z hlediska ovládání co nejjednodušší. Subjektivně je velmi podstatný grafický design. Už samotná ikona aplikace na ploše mobilního zařízení může vzbudit nedůvěru, odstup nebo naopak pocit prospěšnosti či nepostradatelnosti. Uživatele může přesvědčit také tzv. „unfair advantage“, jinými slovy výjimečná výhoda oproti konkurenci. Zmíněný Shazam umí například automaticky vyhledávat hudbu, která hraje v okolí, aniž by musel uživatel explicitně provádět dodatečné akce a získává tak výhodu oproti konkurenční aplikaci SoundHound.

Významnou roli hraje i izolovanost aplikací. Například zmíněná aplikace Shazam umí hudbu nejen najít ale také ji zakoupit nebo předat do jiné aplikace, ve

které si lze hudbu poslechnout. Interakce mezi aplikacemi může mít vliv i v případě plateb. Nejen, že si uživatel prostřednictvím aplikace objedná produkt, ale například pomocí jiné spolupracující aplikace může ihned provést platbu a nákup dokončit.

Z hlediska ambient intelligence je důležitá interakce zařízení s okolním světem a využití moderních technologií za účelem předvídat požadavky uživatele. Aplikace CG Transit pro iPhone nebo Pubtran pro Android využívají GPS pro získání aktuální polohy a nabídnutí nejbližší zastávky. Je-li uživatel zvyklý s výchozí zastávky jezdit na stále stejnou konečnou zastávku, aplikace může přímo předvyplnit cílovou stanici. K tomu všemu je však třeba mít aktuální data a spolupracovat s aplikačním rozhraním partnerů. Moderní aplikace je ale schopna data i uchovávat a v případě ztráty konektivity tato data poskytnout. I bez připojení k internetu tak může uživatel zjistit, jak se nejsnáze dostane mezi zvolenými zastávkami.

Každá aplikace by měla respektovat soukromí uživatele a bezpečnost jeho dat. Aplikace poskytující informace o zakoupené letence by neměla bez prokázání identity nabídnout možnost zrušení letů nebo aplikace mobilního bankovnictví by z hlediska bezpečnosti měla požadovat ověření při každém pokusu o použití. Oblastí, které je třeba respektovat je celá řada a vývoj komplexní kvalitní aplikace už v dnešní době často není v silách jednoho jedince. Vývoj probíhá v týmech, ve kterých má každý člen svou specifickou roli v některé z výše zmíněných oblastech.

2.5. Závěry a shrnutí

Vydavatelé mají za cíl vytvoření úspěšné aplikace, která bere na zřetel chování a požadavky uživatele a snaží se je co nejintuitivněji řešit. Prostředkem k tomu je libovolná technologie s vybraným programovacím jazykem použitým nad sadou vývojářských nástrojů SDK. Veškeré technologické zázemí je uživatelem vnímáno v podobě grafického vzhledu, který respektuje požadavky definované grafickým manuálem platformy. Vzhledem k fragmentaci zařízení se zároveň předpokládá, že aplikace je dostatečně otestovaná na různých zařízeních.

3. Návrh metodiky vývoje mobilních aplikací

V první a druhé části práce byly zanalyzovány metodiky pro vývoj software a shrnuty specifické rysy vývoje software pro mobilní zařízení včetně popisu odlišností na jednotlivých platformách. Z druhé části práce vyplývá, že pro tvorbu nativních mobilních aplikací je důležité především grafické rozhraní aplikace, které uživateli přináší vysokou použitelnost a uživatelskou přívětivost. Právě realizaci vizuálního vzhledu a jeho validaci je věnována velká část této kapitoly následována popisem implementace a procesem testování a doručení aplikace.

3.1. Předpoklady metodiky

Na základě předešlých kapitol jsou dále odvozeny následující logické závěry a předpoklady pro vznik metodiky vývoje mobilních aplikací.

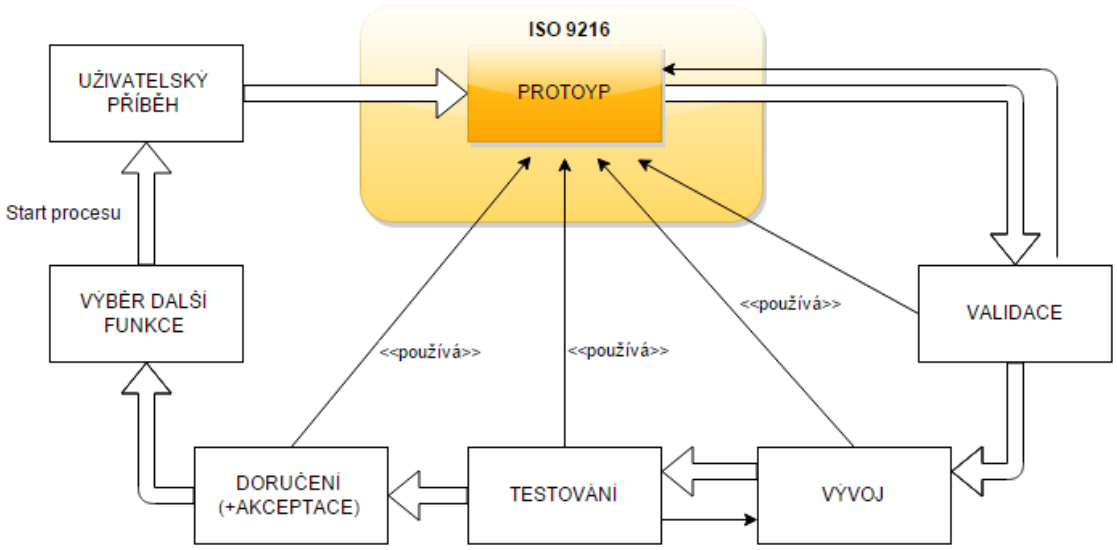
- Pokud agilní projekty mají větší potenciál úspěchu, ať je metodika agilní a splňuje požadavky Agile Manifesto
- Pokud je pro uživatele klíčový především obsah a přístup k němu skrze dobře vyřešené UX, UI a design, ať je tato fáze naplněna prototypem a tento prototyp je klíčovým prvkem metodiky, ze kterého bude každá funkce realizována
- Pokud se osvědčuje spolupráce se zákazníkem, ať je tato spolupráce v metodice preferována
- Pokud může být potenciálním problémem metodik značné množství rolí, ať je jejich počet minimalizován
- Pokud Continuous Delivery snižuje stres a přináší větší jistotu při nasazování nových verzí aplikace, ať metodika vychází z neustálého doručování nových funkcí
- Pokud se požadavky na systém stále mění, ať má možnost zákazník měnit priority neustále
- Pokud se osvědčují jednotkové testy jako prostředek ke snížení počtu chyb, ať jsou testy jednotek volitelně zahrnuty do metodiky a procesu testování

- Pokud je prototyp výchozím prvkem metodiky, ovlivňuje další proces vývoje a je nutná jeho bezchybnost, ať má prostředek, kterým bude validován
- Pokud se osvědčuje testování jako praktika k zamezení vzniku chyb, ať je důkladně testována či ověřována každá fáze procesu vývoje
- Pokud chyby vytváří náklady a zvyšují nedůvěru uživatelů ve vydavatele, ať jsou chyby odstraněny vždy co nejdříve a jejich vyřešení je upřednostněno před vývojem a distribucí nových funkcí

Vzhledem k tomu, že metodika vychází z prototypu a jeho validace na straně dodavatele, není metodika navržena pro případy, kdy zadavatel dodává hotovou specifikaci. Ačkoliv metodika doporučuje v dalších částech validaci prototypu odborníky v oblasti návrhu, může tuto validaci provést s cílovou skupinou i přímo zadavatel (produktový vlastník).

3.2. Funkční rámec

Základním východiskem metodiky je sada uživatelských příběhů získaných od produktového vlastníka. Každý uživatelský příběh bez ohledu na to, zda je cílem opravení chyby či vytvoření funkce musí projít několika fázemi výroby. Vzhledem k důležitosti vizuálního vzhledu aplikace je uživatelský příběh výchozím prvkem pro vznik prototypu. Ten je klíčovým krokem při tvorbě nových funkcí aplikace a je nutné věnovat značné úsilí jeho validaci. Následně může být prototyp implementován a přeměněn v reálnou funkci. I tato reálná funkce by měla být důkladně otestována. V případě, že se při realizaci jakékoliv fáze procesu objeví nečekaná chyba, kterou není možné v rámci fáze řešit, pak je nutné, aby se uživatelský příběh vracel zpětně k úpravě. Poté však musí opět prostoupit všemi fázemi vývoje. Testování by mělo být intenzivnější v momentě dokončení nové funkce před reálnou publikací na virtuální tržiště. Základní funkční rámec popisuje následující obrázek:



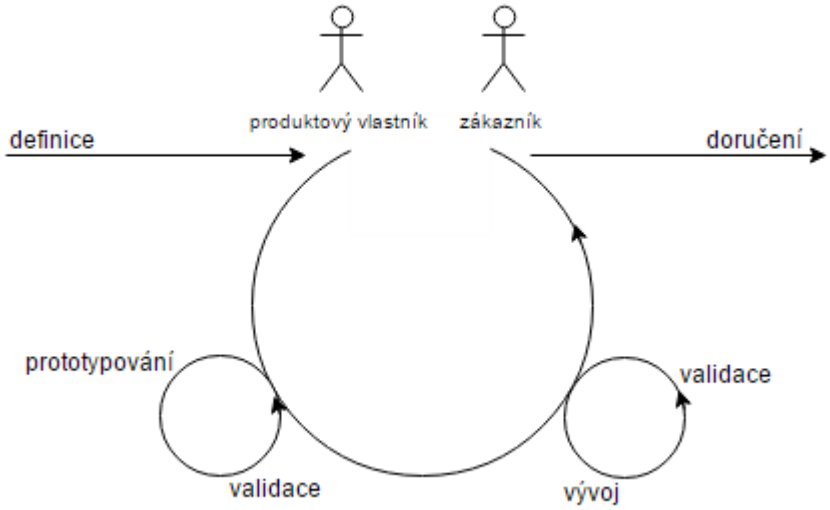
Obr. 3: Rámec metodiky PDMD, zdroj: autor

Na obrázku je naznačen proces Continuous Delivery, který je vedle tvorby prototypů důležitou praktikou celé metodiky. Vzhledem k významu prototypu využívaného během celého procesu vývoje, je dále tato metodika označena jako *„Prototypem řízený vývoj mobilních aplikací“*, anglicky též *„Prototype driven mobile development“*, zkr. *PDMD*. Protože tento prototyp řídí proces validace po celou dobu vzniku nové funkce, je tento proces validace dále označován jako *„Prototypem řízená validace“*, anglicky též *„Prototype driven validation“*.

3.2.1. Continuous Delivery

Požadavky v podobě uživatelských příběhů jsou shromažďovány a ve většině metodik řešeny v různě dlouhých iteracích, na jejichž konci je aplikace doručována zákazníkovi nebo na virtuální tržiště. Takto vytvářené iterace mají však pro mobilní vývoj řadu nevýhod. Vzhledem k vymezení iterace časem nelze zaručit, že na jejím konci bude vzniklý produkt konzistentní. V případě delších iterací se nahromadí řada nových funkcí a ty mohou uživatele zahltit. Pokud vzniknou neočekávané chyby v produkci, je nutné se vracet ke starším verzím, které obvykle uživateli stejně velkou množinu funkcí seberou a mohou ho tím zmást. V neposlední řadě zopakujme fakt, že mobilní aplikace je typu tlustý klient a tudíž doručení nové

verze může trvat i několik dní. Je-li uživatel konzervativní, pak i přes nově vydanou verzi není zaručeno, že si tuto verzi okamžitě stáhne. (9)



Obr. 4: Pohled na proces Continuous Delivery

Vzhledem k výše zmíněnému dává smysl publikovat každou stabilní verzi aplikace, která přináší novou funkci a zavedení iterací použít pouze v případě integrace s dalšími agilními metodikami. Aby proces Continuous Delivery fungoval správným způsobem, je vhodné nastavit určitá rámcová pravidla:

- Musí být zajištěna neustálá prioritizace uživatelských příběhů
- Uživatelské příběhy musí být realizovány na základě priority
- Přednostně musí být vyřešeny chyby (defekty)
- Každý uživatelský příběh s validním prototypem musí být dokončen
- Každý realizovaný uživatelský příběh (otestovaná funkce) musí být ihned doručen

Na základě těchto pravidel lze zajistit neustálé doručování nových funkcí, prioritní řešení vzniklých chyb a zabránění neustálého přepínání mezi řešenými uživatelskými příběhy. Tyto principy musí být upřednostněny bez ohledu na integraci s jinými metodikami.

3.3. Analýza a definice aplikace

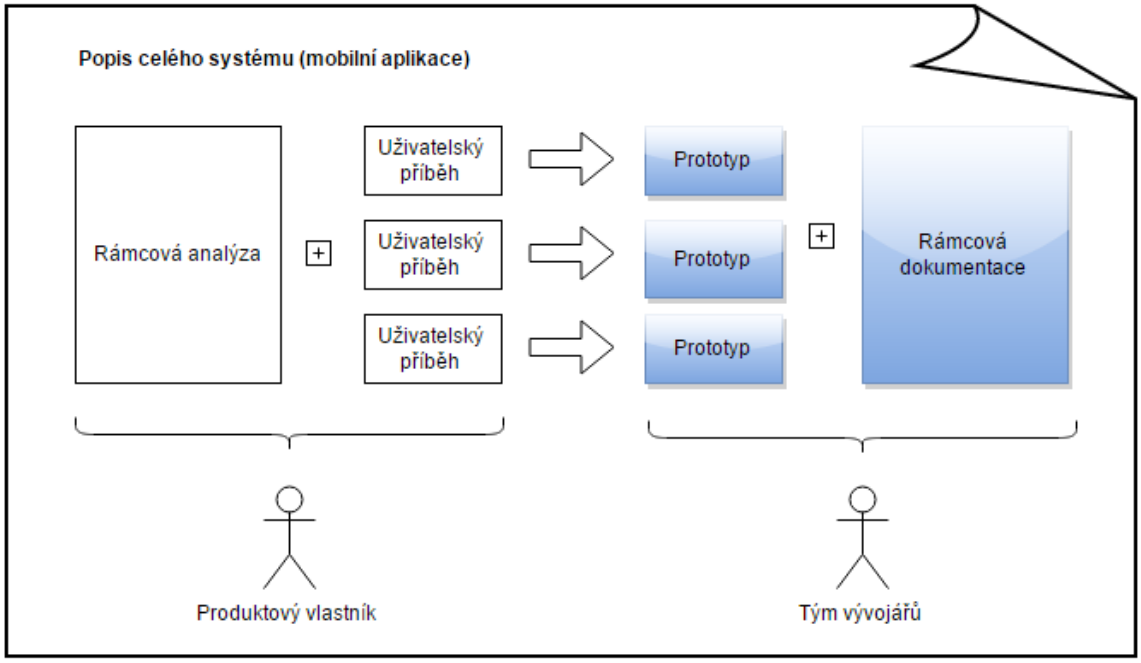
Každá metodika shromažďuje požadavky na funkcionalitu aplikace a tyto požadavky přeměňuje v produkt. Mobilní aplikace se vyznačují tím, že funkcionality jsou „menší“ než v případě desktopových či webových aplikací. Téměř každá funkcionalita je možná popsat uživatelským příběhem stejně jako je tomu v metodikách XP nebo například SCRUM a takový popis dostatečně funkci vystihuje.

Pro představu o aplikaci jako celku je vhodné mít však také rámcovou analýzu doplněnou o funkční i nefunkční požadavky, ze kterých je možné v průběhu vývoje vycházet. Vhodné je, aby taková analýza byla dynamická a s vývojem produktu se také měnila. Součástí rámcové analýzy by měla být i vize produktu včetně přínosu pro cílovou skupinu. Rámcová analýza by měla odrážet představy vlastníka produktu, tedy zákazníka vývojového týmu. Zároveň by neměla být nadbytečně dlouhá, speciálně pokud produktový vlastník s vývojářským týmem úzce spolupracuje. Platí zde pravidlo upřednostnění komunikace se zákazníkem před tvorbou dokumentace a vyjednáváním smluv. Autorem rámcové analýzy by měl být především produktový vlastník společně s osobami, které působí nebo jsou seznámeny s problémovou doménou, typicky vlastníkem produktu.

3.3.1. Dokumentace

Velká část metodik ignoruje tvorbu dokumentace, jelikož know-how je předáváno mezi členy týmu verbálně a dokumentace je dynamicky tvořená za běhu například s využitím Test First přístupu. Jsem přesvědčen, že v agilním vývoji by neměla podrobná dokumentace vznikat před tvorbou funkcí a pokud již ano, neměla by řešit otázku „Jak“ ale především „Co“. Důvodem je především fakt, že způsob řešení by měl být navržen odborníky na danou fázi výroby a ideálně až v moment sesbírání všech specifikací. Vzhledem k tomu, jak se software a technická implementace neustále dynamicky mění je vhodné energii na vznik dokumentace věnovat jiným fázím. Popisem produktu pak mohou být vznikající a dokončené uživatelské příběhy fungující nad rámcovou analýzou a především pak jejich realizace v podobě prototypů.

Vedle rámcové analýzy však dává smysl retrospektivně dokumentovat hotové funkce, které vystihují podstatu celého řešení. Zatímco rámcová analýza odráží produkt z pohledu zákazníka, rámcová dokumentace odráží technické řešení aplikace z pohledu vývojářského týmu. Součástí rámcové dokumentace by měl být popis použitých technologií, popis klíčových částí systémů nebo nastínění architektury aplikace. V případě komunikace s webovými službami nebo dalšími systémy by mělo být do dokumentace zahrnuto schéma komunikace. Nástrojem rámcové dokumentace může být slovo i komplexní sada UML diagramů a jejími autory by měli být zodpovědné osoby uvnitř vývojářského týmu.



Obr. 5: Vznik dokumentace, zdroj: autor

3.4. Formalizace požadavků

Trendem současných metodik je popis požadavků pomocí uživatelských příběhů. Jejich výhodou je, že popisují pouze „Co“ se má vyvíjet, ale již neukládají „Jak“ se má daný příběh realizovat. Vzhledem k tomu, že podoba funkce je nejistá do posledního okamžiku, jeví se uživatelské příběhy jako ideální výchozí stavební blok

nové funkce. Nespornou výhodou uživatelských příběhů je i fakt, že pokud se změní způsob řešení určité funkcionality, uživatelský příběh zůstává nezměněn.

3.4.1. Produktový vlastník

Za účelem efektivní komunikace by měl být tvůrcem uživatelským příběhů klient reprezentovaný jednou osobou. Tato osoba je v této práci dále označována jako produktový vlastník. Role produktového vlastníka by měla být velmi podobná téže roli v metodice SCRUM. Cílem produktového vlastníka je stanovit vizi vznikajícího projektu a sestavit rámcovou analýzu. Vývojový tým na základě této analýzy musí získat představu o produktu jako celku, o jeho cílové skupině a pochopit cíle, které má produkt naplnit. Součástí analýzy by měl být i výčet externích systémů, se kterými aplikace komunikuje, jejich předpokládaná role a další informace, které mohou pomoci vývojářskému týmu navrhnout architekturu aplikace.

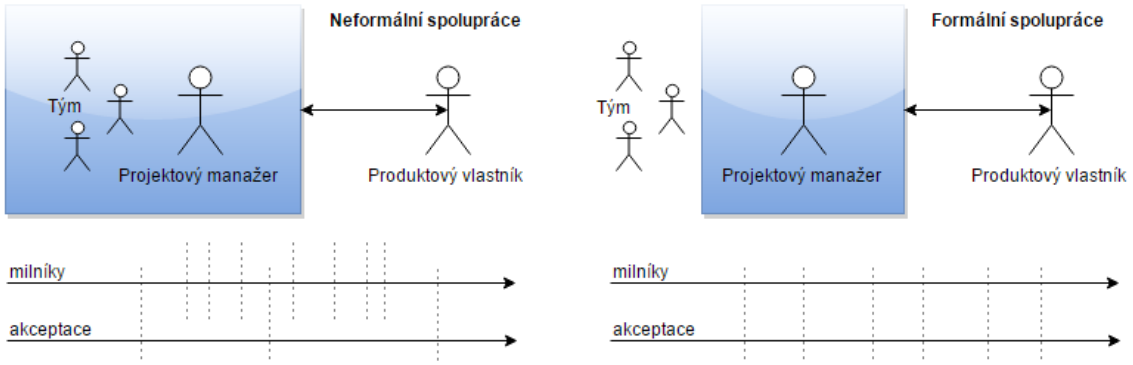
Dalším klíčovým úkolem produktového vlastníka je vytvářet uživatelské příběhy na základě komunikace s uživateli a tyto uživatelské příběhy předávat vývojářskému týmu skrze projektového manažera. Produktový vlastník dále mění prioritu uživatelských příběhů a hotové příběhy v podobě funkcí předává svému vnitřnímu týmu k akceptaci.

3.4.2. Projektový manažer

Vývojářský tým potřebuje komunikovat se zákazníkem za účelem realizace uživatelských příběhů. V případě formálnější spolupráce lze zvolit zástupce týmu, který je zodpovědný za řízení jednoho konkrétního projektu. Dále tuto osobu budeme nazývat projektový manažer. Projektový manažer by měl sbírat požadavky od produktového vlastníka a předávat je vývojářskému týmu k vyřešení. Metodika může velmi dobře fungovat i za předpokladu, že role projektového manažera bude upravena do role Scrum Mastera, podobně jako v metodice SCRUM. Lze tak vytvořit na straně dodavatele méně formální a motivující prostředí pro vývoj aplikací.

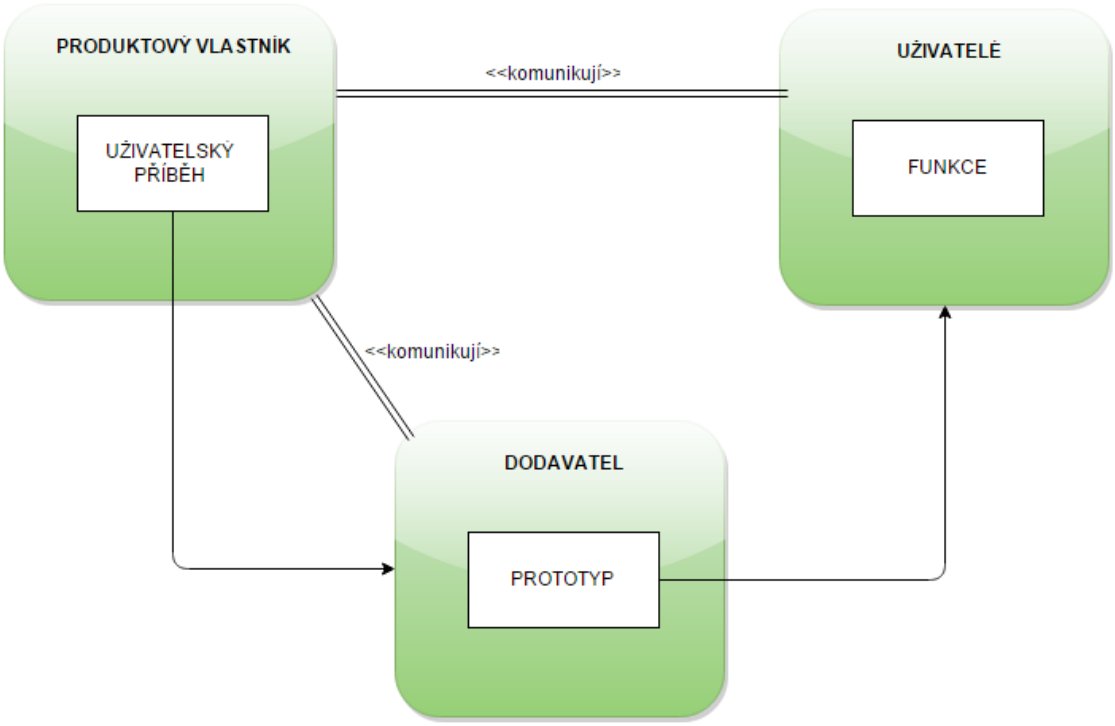
3.4.3. Formální a neformální spolupráce

Z popisu výše vyplývá, že spolupráce mezi projektovým manažerem a produktovým vlastníkem může být buď těsná a formální nebo volná a zároveň i méně formální. Neformální spolupráce se přibližuje více k agilnímu vývoji aplikací a lze ji doporučit především v případě menších projektů. Pro rozsáhlé systémy je však vhodnější stanovit jednu zodpovědnou osobu za dodavatele.



Obr. 6: Popis formální a neformální spolupráce, zdroj: autor

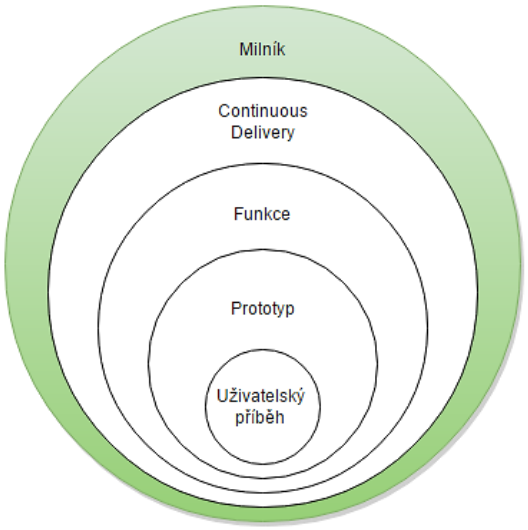
Z obrázku vyplývá, že v případě neformální spolupráce se tým s produktovým vlastníkem schází častěji, panuje mezi nimi větší důvěra a probíhá méně akceptačních procesů. V případě formální těsné spolupráce se schůzky nekonají tak často avšak množiny distribuovaných funkcí jsou častěji testovány zákazníkem v podobě akceptačních testů. Celkový pohled na proces dodávání nových funkcí včetně rolí v systému znázorňuje obrázek č. 7.



Obr. 7: Proces realizace uživatelského příběhu, zdroj: autor

Milníky

Milníky jsou momenty, které stanoví produktový vlastník po dohodě s projektovým manažerem. Protože agilní metodiky pracují s fixním časem, i milník je vhodné stanovit časem a předpokládat, že v době milníku bude realizována část uživatelských příběhů. Cílem milníků je především retrospektiva nad celkovým produktem a stanovení další vize a vývoje. Během této retrospektivy se zároveň aktualizuje rámcová analýza aplikace. Na rozdíl od iterací a sprintů, které definují jiné metodiky, milníky mohou být nepravidelné. Výhodou takového přístupu je oproštění od závazku obou stran, scházet se na pravidelných schůzkách s pevnou periodou, kdy reálná potřeba schůzek je větší či menší. Metodika nicméně nebrání tomu, aby se milníky konaly pravidelně a vytvořily tak iterační běhy. Na konci milníku není nutné vydání nová verze aplikace, jelikož doručování nových funkcí probíhá neustále v rámci nezávislého procesu Continuous Delivery.



Obr. 8: Náhled hierarchie artefaktů metodiky, zdroj: autor

3.4.4. Tvorba uživatelských příběhů a správa backlogu

Uživatelský příběh je neformální popis požadavku dodaný produktovým vlastníkem, který má být realizován dodavatelem. Konečným výstupem provedeného příběhu je nová funkce aplikace (ať už se jedná o grafické nebo technické vylepšení). Uživatelské příběhy mají celou řadu formátů a pro metodiku není klíčová jejich podoba. Důležitá je odpověď na 3 základní otázky:

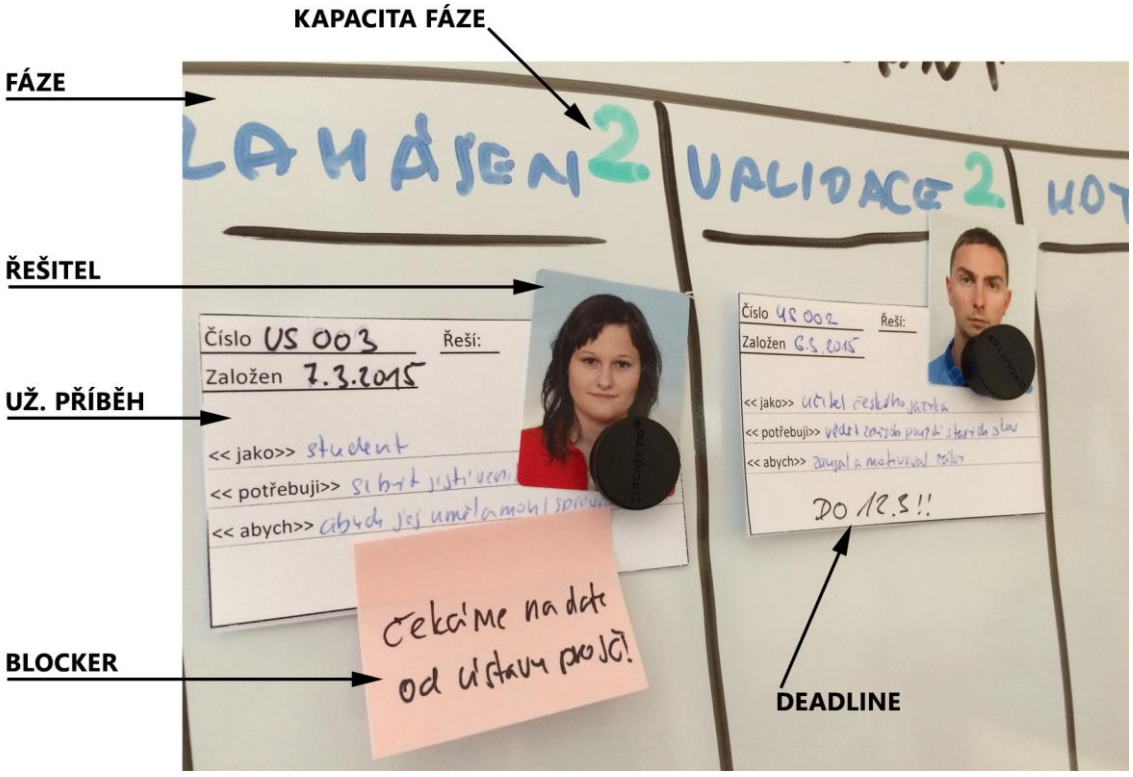
1. Pro koho je nová funkce vytvářená (role)
2. Jaká je potřeba této role (popis funkce)
3. Jaký je přínos vytvořené funkce pro danou roli

Příklady uživatelských příběhů mohou být následující:

- Jako <role> chci <funkci> aby <přínos>
- Jako <role> chci <funkci>
- Abych dostal <přínos> jako <role> chci <funkci>

Příkladem uživatelského příběhu může být požadavek na vyhledávání telefonního čísla v mobilní aplikaci: *„Aby bylo možné co nejrychleji vyhledat telefonní*

číslo, chce uživatel, aby na hlavní straně bylo dostupné vyhledávací pole s našeptáváním.“. Příběh by nikdy neměl naznačovat, jakým způsobem má být funkce vytvořena (dále ozn. Spoiler), ale pouze co by mělo být vyvíjeno, pro koho je daná funkce vytvářena a jaká je její podstata nebo přínos pro uživatele. Uživatelský příběh může být dále doplněn o popis zpřesňující požadavky nebo další postřehy. Každý uživatelský příběh by měl být dále vhodně označen datem návrhu, unikátním číslem a zodpovědnými osobami, které v dané fázi uživatelský příběh realizují. V případě většího množství uživatelských příběhů je vhodné uvést i prioritu. (18)



Obr. 9: Ukázka uživatelského příběhu, zdroj: autor

Blockers (obecně úkoly)

Přestože celá řada agilních metodik využívá rozpad uživatelských příběhů na úkoly, vhodnější je dekompozice komplikovaných uživatelských příběhů na menší uživatelské příběhy. Metodika má za úkol řídit především tok potřeb uživatele s cílem přeměny na fyzické funkce. Je-li nutné řídit i úkoly, pak je vhodné

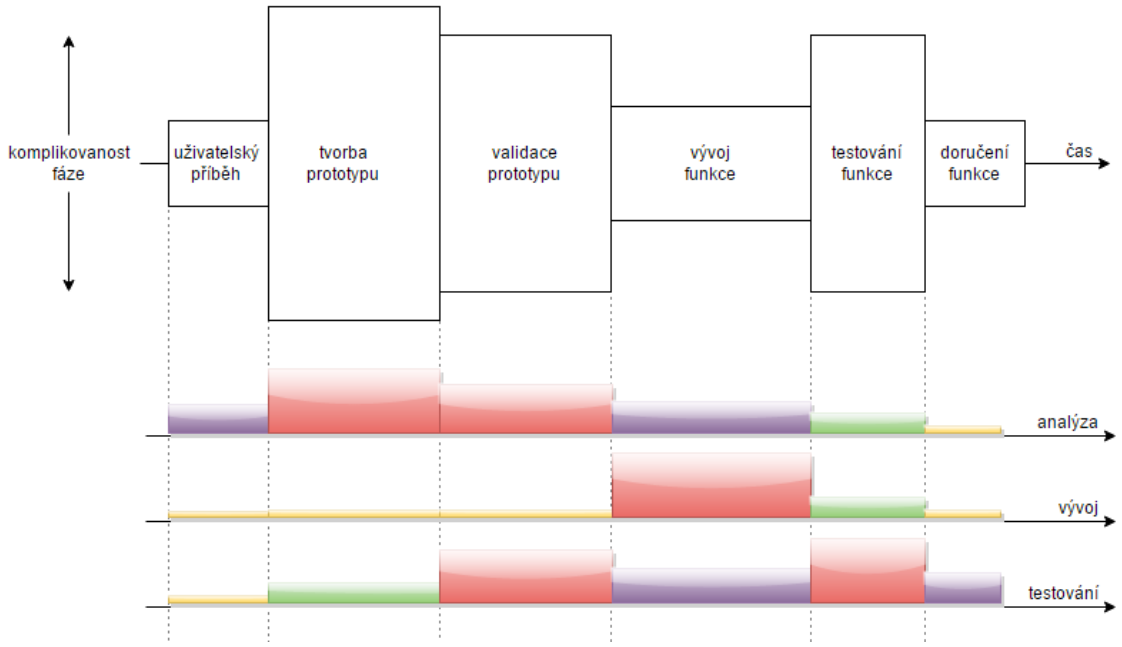
tyto úkoly přímo spojit s uživatelským příběhem. Splnění úkolů obvykle blokuje splnění celého uživatelského příběhu (odtud Blockers). Proto je nutné úkoly vyřešit co nejdříve.

Výchozí uživatelský příběh

Protože rámcová analýza obvykle klade základní požadavky na projekt, bývá na základě této analýzy možné vytvořit návrh architektury systému. Ihned po dodání rámcové analýzy je vhodné vytvořit výchozí (nultý) uživatelský příběh a na základě tohoto příběhu navrhnout základní prvky systému. Při realizaci výchozího příběhu je vhodné implementovat především architekturu aplikace nebo například ověřit možnosti komunikace s externími službami. Tak, jako rámcová analýza vytváří představu o celkovém vyvíjeném produktu, měla by realizace výchozího příběhu do určité míry řešit technické řešení projektu známé ze studie proveditelnosti. Výchozí příběh by měl být realizován samostatně a teprve po jeho dokončení by mělo být zahájeno provádění dalších uživatelských příběhů. Výstupem tohoto příběhu není obvykle funkce ale architektonický návrh a upozornění na rizika a možnosti jejich řešení. Během realizace výchozího uživatelského příběhu vzniká logicky i zásadní část rámcové dokumentace.

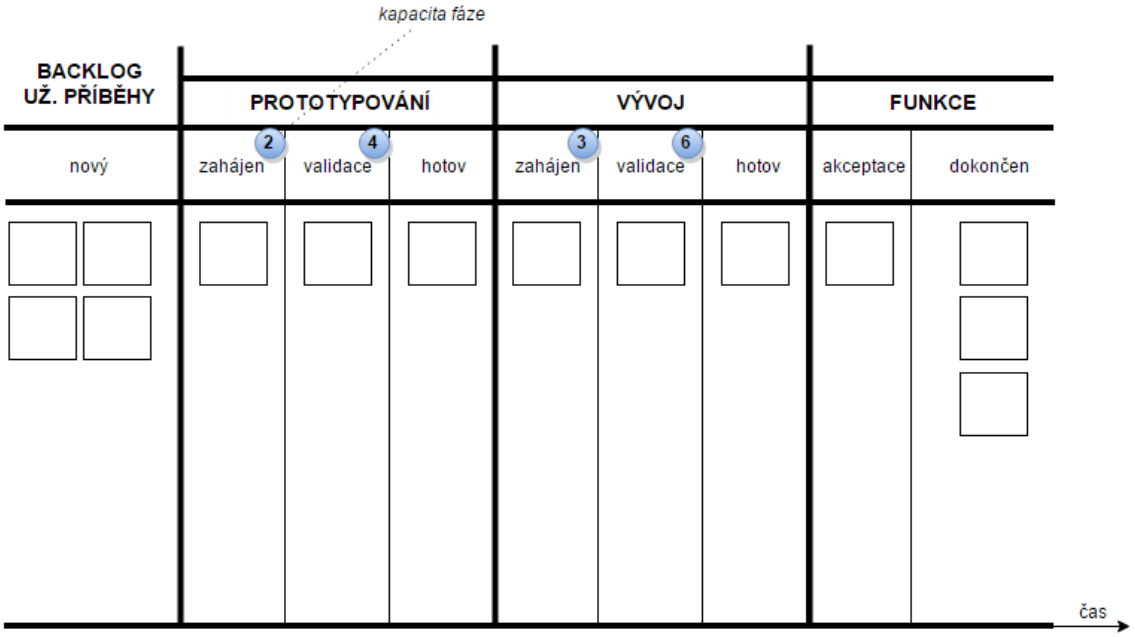
Životní cyklus uživatelského příběhu

Každý uživatelský příběh prochází několika fázemi a reflektuje stav výroby dané funkcionality. Fáze vývoje každé funkce lze nastínit diagramem na základě funkčního rámce definovaného v kapitole 3.2 a zároveň jej doplnit i o odpovídající stav uživatelského příběhu.



Obr. 10: Fáze vývoje aplikace a stavy uživatelského příběhu, zdroj: autor

Se známými stavy lze dále vytvořit týmovou tabulku, do které lze uživatelské příběhy vkládat, posouvat je v rámci fází a měnit jim zodpovědnou osobu. Tato tabulka vychází z doposud nezmíněné metody Kanban a na rozdíl od metodiky Scrum, která definuje kapacitu v rámci sprintů, metodika PDMD limituje počet uživatelských příběhů v rámci výrobních fází. Tabulka je dále v této práci označována jako Kanban Board.



Obr. 11: Kanban board, zdroj: autor dle agileproductdesign.com

Paralelní realizace více scénářů

Uživatelské příběhy mohou být prováděny paralelně podle kapacity týmu s jedinou výjimkou výchozího uživatelského příběhu. Cílem paralelizace je, aby byl tým neustále pokrytý prací. Dále dává smysl slučovat vývoj prioritních uživatelských příběhů, které spolu souvisí, čímž je možné ušetřit mnoho času. Speciálně ve fázi validace je vhodné nashromáždit více uživatelských příběhů a ty pak validovat na jedné uživatelské skupině hromadně. Při paralelizaci uživatelských příběhů je nezbytné nastavit definovanou kapacitu každé fáze a tato kapacita následně udává maximální počet řešených uživatelských příběhů v daný čas v dané fázi.

Backlog

Cílem backlogu je shromažďovat a vést detailní přehled o funkcích systému v podobě uživatelských příběhů. Produktový vlastník určuje prioritu uživatelských příběhů a dodavatel tyto příběhy přetváří v rámci procesu na nové funkce systému. Backlog je určen k vedení uživatelských příběhů i chyb, které jsou jejich speciálním případem. Backlog tabulka může být rozšířena i o časové odhady, čímž lze získat

představu o době realizace jednotlivých funkcí a plánovat snadněji milníky. Backlog nabízí ve své podstatě to samé jako Kanban board s tím rozdílem, že obsahuje i aktuálně neřešené uživatelské příběhy a doplňuje je o další údaje užitečné pro plánování v rámci milníků.

Tab. 1: Backlog, zdroj: autor

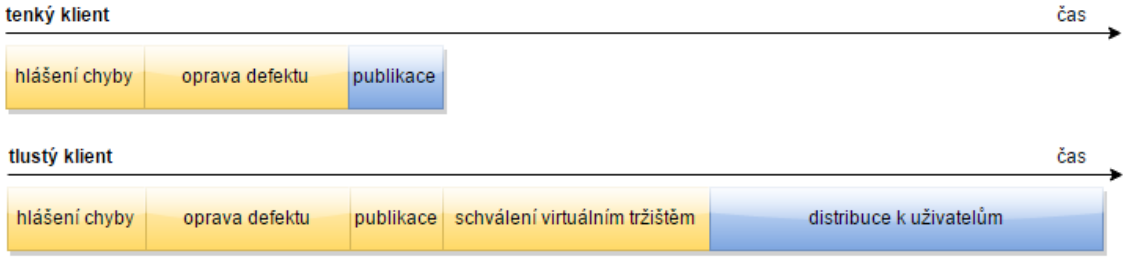
DEF 005	<stav>	<popis DEF>	<datum založení>	<odp. osoba>
US 009	<stav>	<popis US>	<datum založení>	<odp. osoba>
US 005	<stav>	<popis US>	<datum založení>	<odp. osoba>

Defekt

Každá nahlášená chyba je označena jako defekt. Technicky je defekt velmi podobný uživatelským příběhům ale má svá specifika související s popisem. Tato specifika mají za cíl usnadnit identifikaci problému. Hlášený defekt by měl být definován dvěma větami v podobě očekávaného chování a skutečného chování. Defekt lze dále doplnit o poznámky, připojené screenshoty nebo fotografie doplňující popis vzniklé chyby. Součástí hlášeného defektu by stejně jako v případě uživatelského příběhu neměl být návrh řešení (spoiler). A to ani v případě, že uživatel má představu, kde by mohla být příčina chyby. Takový spoiler může totiž mylně navést dodavatele jinam než k jádru problému a prodloužit dobu řešení defektu. Defekty je vhodné na Kanban boardu odlišovat, například štítkem jiné barvy. V Backlogu se defekty analogicky zařadí zcela na vrchol.

Prevence defektu

Při vývoji mobilních aplikací je vznik defektu velmi nežádoucí jev. Jak bylo nastíněno v kapitole 2.1.1, defekt vzniklý při reálném používání produktu nelze odstranit zdaleka tak snadno, jako v případě tenkého klienta. Ačkoliv vyřešení defektu může být rychlé, jeho promítnutí k uživateli může trvat několikanásobně déle.



Obr. 12: Rozdíl doby opravy defektu, zdroj: autor

Na obrázku je popsán rozdíl mezi vyřešením defektu v případě tenkého a tlustého klienta. Jiným problémem může být, že zatímco publikace opravené verze v případě tenkého klienta se dotkne všech uživatelů, distribuce opravené verze na tlustém klientovi je teoreticky nekonečné dlouhá, jelikož uživatel musí sám nainstalovat novou verzi aplikace. Právě z těchto důvodů je důležité chyby nejen řešit ale především i najít prostředek, kterým se zamezí opakovanému vzniku produkčních chyb.

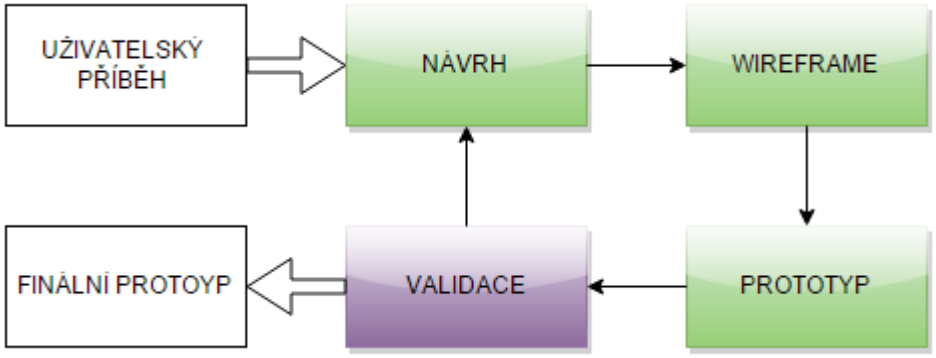
3.5. Prototyp aplikace

První fází po vytvoření rámcové analýzy a získání pohledu na vytvářenou funkci je její vizuální reprezentace. Tento krok je prioritní především proto, že vizuální vzhled aplikace je klíčový pro uživatele. První co uživatel při interakci s mobilní aplikací vnímá je její vzhled. Poprvé při výběru aplikace na virtuálním tržišti, následně ikonu aplikace před jejím prvním spuštěním a nakonec aplikaci samotnou během jejího používání. Uživatel nevidí, nevnímá a nezajímá ho, zda aplikace byla vytvořena s využitím supermoderních technologií, zda kód aplikace je dokonale zrefaktorovaný a čistý a nezajímá ho ani, zda použité návrhové vzory jsou efektivní či nikoliv (ačkoliv pro produkt se jedná o důležité vlastnosti). Mezi klíčové oblasti návrhu vzhledu patří rozmístění prvků a obsahu, uživatelský prožitek a následně samotný vzhled řešení. Obsah a data mají výjimečnou roli, protože jsou obvykle důvodem, proč uživatel aplikaci používá. Vizuální vzhled v podobě prototypu tedy reprezentuje kompletní popis požadované funkcionality, vychází z uživatelského příběhu a měl by být nejdůkladněji propracovanou fází. Zatímco

uživatelské příběhy se soustředí na popis funkcionality z hlediska produktového vlastníka, prototyp vytvořený odborníky z oboru již určuje, co vše by měla vytvářená funkce umět a jaké požadavky musí být splněny za účelem naplnění uživatelských potřeb. Další text této kapitoly vychází ze zdrojů (20) a (21).

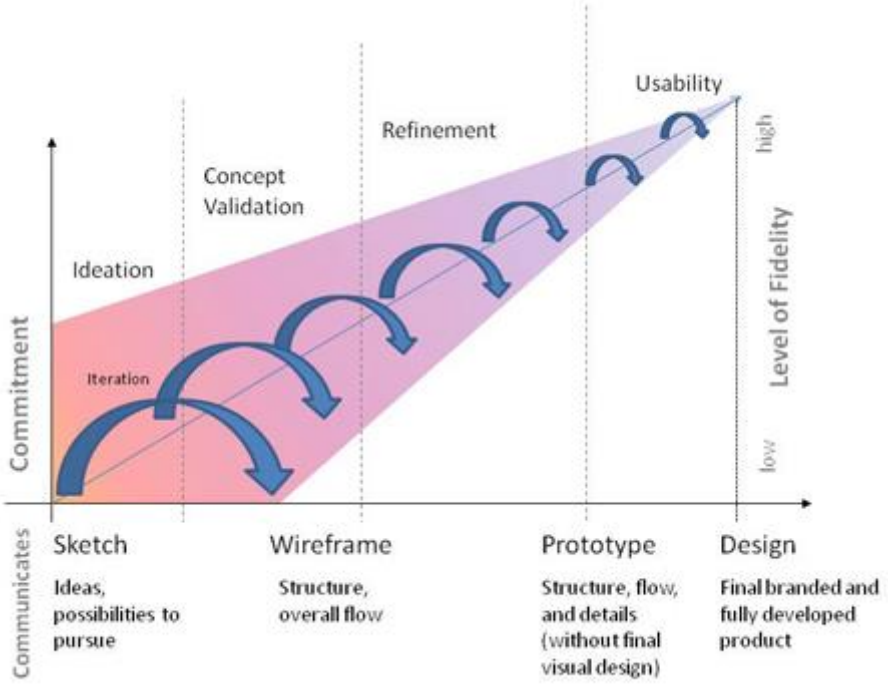
3.5.1. Realizace

Z uživatelského příběhu je cílem vytvořit co nejrychleji vizuální pohled na vytvářenou funkci. Obvykle se jedná o rychlou skicu, kterou dále označme jako návrh. Návrh je velmi rychlý a levný. Jeho cílem není definovat vzhled aplikace, ale především podpořit následný proces prototypování a tvorbu wireframe. Ten je již na testovatelné úrovni a slouží s využitím UX a UI k vytvoření prototypu.



Obr. 13: Proces realizace prototypu, zdroj: autor

Prototyp vylepšuje základní koncept a popisuje záměry, které byly doposud pouze myšlenkou, a dále odpovídá na otázky, které vznikly v průběhu modelování. Výsledkem prototypu jsou online klikatelné dokumenty, vizuální grafické návrhy i vytištěné podrobné návrhy stránek včetně popisu interakce a další specifikace. Dokončený prototyp by měl respektovat UI a UX pravidla, která jsou dále popsána.



Obr. 14: Realizace prototypu a obraz přesnosti, zdroj: uxmatters.com

User Experience

Oblast UX řeší především problém interakce mezi uživatelem a IT systémem. Součástí UX je i vizuální vzhled, běžně označovaný jako grafický design. Při návrhu UX vzniká také tvorba obsahu a informační architektura, podporující dohledatelnost a použitelnost. Součástí použitelnosti je i oblast přístupnosti a zaměření na uživatele s handicapem. V rámci mobilních aplikací nás například zajímá, jaké informace jsou součástí každé stránky, jak má probíhat interakce mezi těmito stránkami a dále co má být součástí kterého menu nebo jaké prvky zobrazit na vybraných místech, aby byla aplikace maximálně použitelná.

Rozsáhlou oblastí UX je interakční design, jehož zodpovědností je naplnit očekávání uživatele, tedy reagovat vhodným způsobem na veškerou akci uživatele. Cílem designéra je vytvoření layoutu aplikace a skrze komunikaci s uživatelem definovat chování aplikace a vybrat pro uživatele nejdůležitější informace a funkce.

User Interface

UI se nachází mezi uživatelem a systémem a jeho cílem je, aby uživatel vyvinul minimální úsilí pro dosažení požadovaného cíle nebo výstupu. Mezi úkoly UI patří zajištění jednoznačnosti v případě, že chce uživatel provést určitou akci, tvorba jednosmyslné ikonografie i naplnění principu responsivity a správného zobrazení v případě displejů s různou úhlopříčkou. UI zajišťuje vznik konzistentního rozhraní, šetří uživateli čas při práci s aplikací a snaží se do určité míry předpokládat i neočekávané chování uživatele.

UX a UI jsou fáze, které často vznikají současně a jsou také fázemi, v rámci kterých je navrženo použití specifických technologií nebo senzorů za účelem vylepšení interakce mezi uživatelem a mobilním zařízením. Výstupem práce designéra je jednoduchá dokumentace popisující obsah a náplň jednotlivých stránek, myšlenkové mapy a mapy interakce mezi stránkami mobilní aplikace. Jako vizuální pomůcku lze použit drátěné modely (wireframes) nebo pro nastínění animací prototypy vycházející z wireframe. Velmi obvyklé je i vytvoření grafického mockupu. Dokumentace by měla být strukturovaná, aby bylo možné každý požadavek testovat.

Grafický Design

Grafický design je oblast, v rámci které jsou výstupy z UX a UI přeměňovány ve fyzický vzhled aplikace, který je nadále realizovatelný v dalších fázích procesu vývoje. V procesu vzniku reálného designu jsou voleny konkrétní barvy, konkrétní piktogramy, konkrétní typografie a jsou naplňovány požadavky definované v UX. Grafický design v konečném důsledku vytváří konečnou identitu aplikace s využitím zvyklostí vybrané platformy a úkolem grafického designera je rovněž tvorba výstupu v podobě grafických souborů, které je dále možné začlenit do vznikající aplikace. Grafický design by měl vzniknout pokaždé, pokud je to možné, aby nepřešlo rozhodování o vzhledu do technického procesu vývoje. Grafický design není izolovanou částí procesu prototypování, ale jeho výstup je součástí celého prototypu.

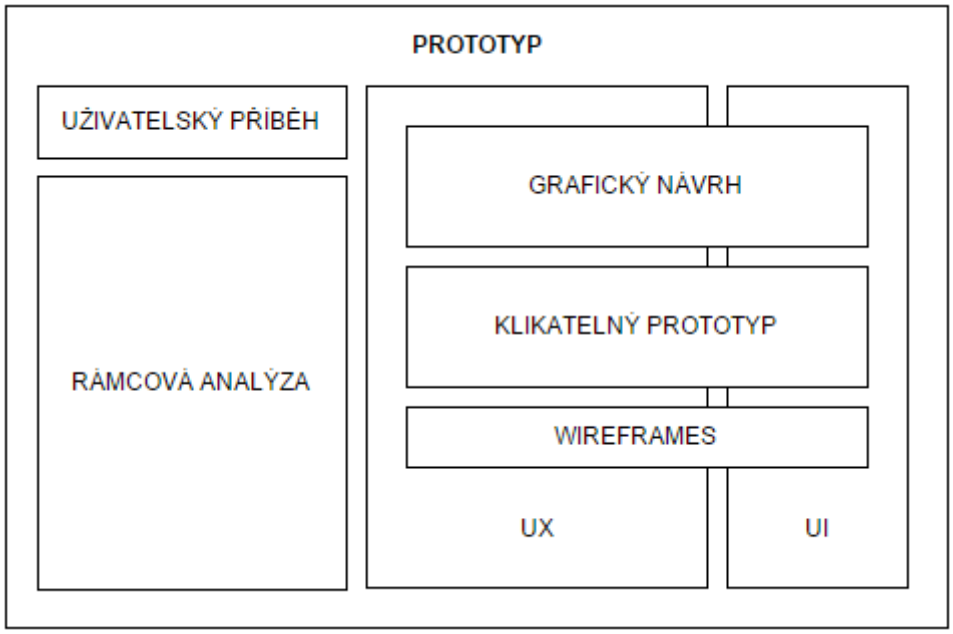
3.5.2. Nástroje

Ve fázi prototypování lze používat celou řadu on-line a off-line nástrojů, které umožňují vytvářet drátěné modely nebo klikatelné prototypy. Nejlepší nástroje dokáží vytvářet detailní prototypy doplněné i o interaktivní chování, díky němuž je možné popsat chování ovládacích prvků a přechody mezi stránkami. Mezi oblíbené nástroje patří Axure RP nebo Solidify.

Kromě sofistikovaných nástrojů lze ale používat i tužku s papírem a vytvářet grafické návrhy v podobě:

- Modelů obsahu,
- Empatických map,
- Storyboardů,
- Scénářů,
- Sitemap,
- Procesních diagramů,
- Skic

Celková podoba prototypu může v konečné fázi zahrnovat všechny výše zmíněné části v kapitole 3.5. Pohled na obsah prototypu poskytuje Obr. 15.



Obr. 15: Součásti prototypu, zdroj: autor

3.6. Validace prototypu

Jak bylo zmíněno v úvodu této kapitoly, zřejmě nevýznamnější fází procesu vývoje mobilní aplikace je vznik prototypu a vizuální reprezentace nové funkce. Aby bylo možné postoupit do technologické fáze procesu vývoje, je nutné ověřit (zvalidovat) samotný prototyp. Protože prototyp již ve své podstatě ukazuje, jakým způsobem aplikace funguje, lze pro jeho validaci použít některý ze softwarových modelů kvality. Modely hodnotí odlišná kritéria a liší se od sebe i svou komplexností. Pohled na vybrané modely kvality je znázorněn v Tab. 2.

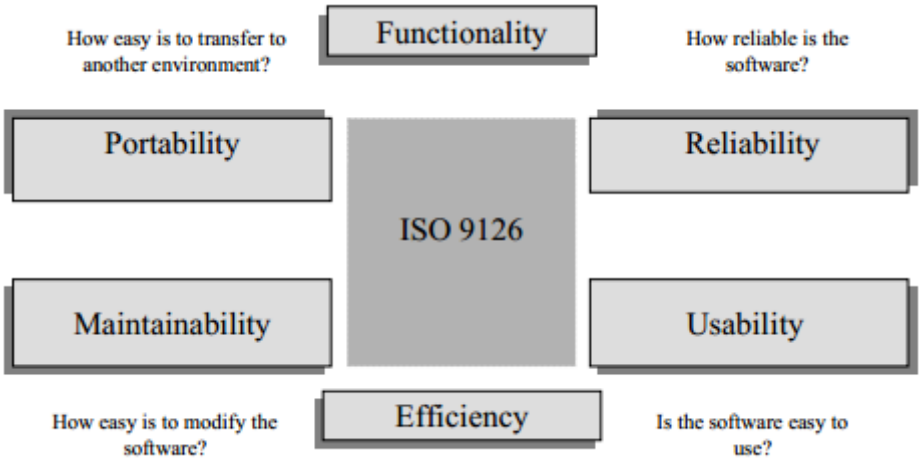
Tab. 2: Modely kvality, zdroj: (20)

Quality Attribute	McCall	Boehm	Dromey	FURPS	ISO 9126
Maintainability	×		×		×
Flexibility	×				
Testability	×	×			
Correctness	×				
Efficiency	×	×	×		×
Reliability	×	×	×	×	×
Integrity	×				
Usability	×		×	×	×
Portability	×	×	×		×
Reusability	×		×		
Interoperability	×				
Understandability	×	×			
Functionality	×		×	×	×
Performance	×			×	
Supportability	×			×	

Protože pro mobilní aplikace je důležitý vizuální vzhled a použitelnost aplikace, je vhodné zvolit méně složitý model, který pokryje právě potřeby validace prototypu. Komplexní McCall je proto vyloučen společně s modelem Boehm. V případě ostatních modelů již nejsou rozdíly tak patrné a pro další text práce související s validací je vycházeno z modelu ISO 9126. Metodiku lze však v tomto bodě rozšířit validací jakýmkoliv komplexnějším modelem. (20) Při popisu procesu validace a modelu kvality ISO 9126 je dále vycházeno ze zdrojů (20) a (22).

3.6.1. Charakteristiky jakosti dle ISO 9126

Model ISO 9126 popisuje několik klíčových atributů, z nichž vyvozuje kvalitu software. Tyto atributy budou nyní podrobně popsány za účelem sestavení požadavků a podkladů pro fázi realizace validace.



Obr. 16: Součásti normy ISO 9126, zdroj: testhuis.com

Funkčnost (functionality)

Vyvíjená aplikace by měla jako celek nebo jako množina souvisejících funkcí splňovat funkční požadavky. Implementace každého případu užití musí z funkčního hlediska odpovídat specifikaci, resp. uživatelským příběhům. Kromě funkční shody je nutné, aby vytvořené řešení bylo výpočetně přesné. Z hlediska využití externích systémů, například aplikačních rozhraní musí správně fungovat výměna dat. V případě chyb je nutné zajistit prostředek pro logování a odesílání zpráv vývojářskému týmu. V neposlední řadě je nezbytné myslet na bezpečnost a soukromí uživatele a s tím související zabezpečení a případně kódování dat.

Použitelnost (usability)

V kapitole 2.4 byla podrobně vysvětlena problematika úspěchu mobilních aplikací a zmíněno bylo, že jedna z důležitých oblastí je použitelnost. V rámci použitelnosti se snažíme vytvořit dostatečně ergonomické, intuitivní a pro uživatele atraktivní řešení. Důležité je, aby byla aplikace pro uživatele srozumitelná a respektovala zkušenosti cílové skupiny s podobnými informačními systémy. Přestože mobilní aplikace obvykle nabízejí jen omezené množství propracovaných funkcí, je nutné, aby se byl uživatel schopen maximum z těchto funkcí snadno a podvědomě naučit.

Účinnost (efficiency)

Účinnost patří v případě mobilních aplikací k významným faktorům. Právě proto, že řada mobilních aplikací je používána v terénu, je vyžadováno, aby co nejrychleji poskytovaly potřebné výsledky. Do této oblasti spadá i schopnost aplikace uchovávat si relevantní data off-line, aby bylo možné s aplikací alespoň částečně pracovat při ztrátě konektivity. Zároveň ukládání dat umožňuje často zkrátit prodlevu mezi požadavkem a odpovědí v případě, že uživatel opakovaně odesílá stejný požadavek. Kromě práce s daty je nutné stejně efektivně využívat i senzory zařízení a pracovat s hardwarovým vybavením.

Bezporuchovost (realiability)

Řešení, se kterým uživatel pracuje, by mělo být dostatečně spolehlivé a bezchybné. Součástí zajištění bezchybnosti je mimo jiné i zajištění detekce chyb a způsob jejich odstranění. V rámci testování by mělo být zajištěno, že nemohou nastat nesprávné operace s aplikací a pokud takové operace nastanou, mělo by být vytvořeno řešení prevence. Příkladem může být snaha uživatele zadat neočekávaný vstup do textového pole. Má-li jím být například telefonní číslo, je nutné nejen zajistit chování aplikace při nečekaném vstupu ale z hlediska UX také uživateli nabídnout adekvátní typ vizuální klávesnice. Pro zajištění a snížení rizika chyb je vhodné budovat funkcionální část aplikace na tenkém klientovi a pro komunikaci využít spojení s tímto klientem. Mezi dvě základní chyby při implementaci vzdálených služeb patří:

1. Přímá komunikace mobilní aplikace s externí službou
2. Komunikace mobilní aplikace s vlastní službou a zpracováním dat na straně tlustého klienta

V prvním případě se mobilní aplikace stává zcela závislou na externím dodavateli. Pokud se změní přístupový klíč k API nebo dojde ke změně kontraktu, pak se stává mobilní aplikace zcela nefunkční a chyba musí být odstraněna na straně tlustého klienta. Druhý případ již první problém částečně řeší tím, že externí služba je využívána vlastním API, avšak data se stále zpracovávají na straně tlustého

klienta. Pokud vznikne požadavek na změnu výpočtů, pak je opět nutná aktualizace mobilních aplikací namísto jedné centrální aktualizace webové služby.

Obr. 17: Popis komunikace se vzdálenými službami, zdroj: autor

Udržovatelnost (maintainability)

Udržovatelnost je schopnost řešení být nadále snadno a za přijatelný čas modifikováno a rozvíjeno. Přestože uživatelé dávají přednost vizuální reprezentaci aplikace a technické řešení je ve své podstatě nezajímá, nemělo by být zcela opomenuto. V kapitole 3.3 byl popsán proces tvorby rámcové analýzy napomáhající

k vytvoření funkční a udržovatelné architektury. Kromě architektonického řešení by měly být i jednotlivé části systému implementovány tak, aby byly stabilní, umožňovaly změnu a především by měly být i snadno testovatelné. Z hlediska testovatelnosti je dobrým předpokladem používání jednotkových testů, byť to není nezbytnou podmínkou. V případě mobilních aplikací lze doporučit využití architektonického vzoru MVVM, který umožňuje simulovat různé vstupy a snadno tak vyvíjenou aplikaci již během implementace testovat.

Přenositelnost (portability)

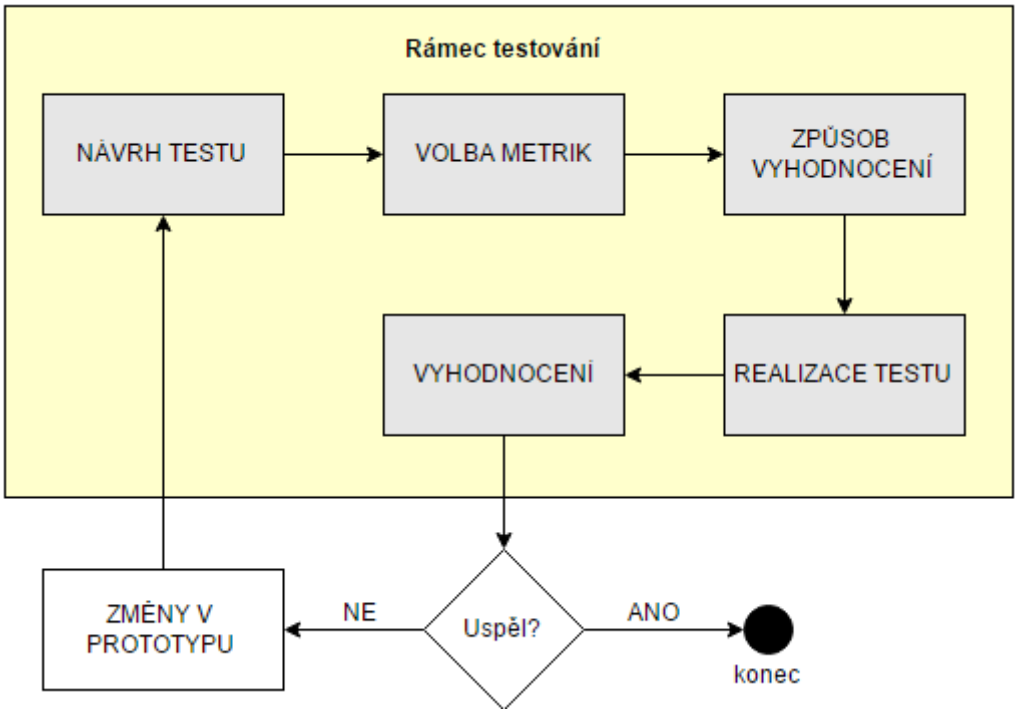
Pro mobilní aplikace je velmi důležitá přenositelnost vyvíjené aplikace. Především platformu Android i Windows Phone trápí problém fragmentace a jelikož mobilní aplikace jsou spuštěné na různých zařízeních, je nutné, aby byly funkční a schopné využívat všech hardwarových prvků všech zařízení. Do této oblasti patří také rozhodnutí, jakým způsobem reagovat, pokud cílové zařízení například nemá požadovaný senzor a v případě výpočetních operací zajistit výkon i na zařízeních s nižším výpočetním výkonem.

Pokud aplikace používá ke své práci jiné, externí systémy nebo aplikace, je potřeba otestovat a zajistit slučitelnost aplikace s každým prostředím, do kterého je aplikace nasazena. Jedná se o tzv. kompatibilitu aplikace s prostředím.

3.6.2. Realizace

Při testování prototypu vycházejícího z uživatelského příběhu je nutné, aby vznikla řada testovacích scénářů ověřujících splnění zcela všech aspektů zvoleného modelu kvality. Scénáře jsou založeny na principu ověřování očekávaného a skutečného chování, schopností dokončit úkoly a sledováním způsobu, jak jsou testovací scénáře uživateli řešeny. Pomocné mohou být i různé textové scénáře, do kterých uživatel doplňuje slova nebo vybírá z předem zvolených možností. Sesbíraná data z různých typů testů se agregují a vyhodnocují proti definovaným předpokladům. Pro tuto fázi je velmi důležité stanovení metrik a způsobu vyhodnocování ještě před samotným provedením testů. Pohled na testování je znázorněn na Obr. 18 a více se této oblasti věnuje mezinárodní standard

ISO/IEC/IEEE 29119. Pro relevantní provedení testů je také nutné dostatečné množství testujících uživatelů. Dle doktora Jakoba Nielsena, věnujícího se problematice použitelnosti, je ideální počet mezi pěti a deseti uživateli. (20) Takto nastavený počet uživatelů je dostatečný pro odhalení většiny potenciálních problémů. Pro oblast mobilních aplikací je kladen na některé atributy modelu kvality větší či menší důraz v závislosti na vyvíjeném řešení. Obecně nejdůležitější aspekty jsou funkčnost a použitelnost.



Obr. 18: Proces testování, zdroj: autor dle ISO 29119

Validace funkčnosti

Při validaci funkčnosti testujeme, zda jsou v mobilní aplikaci přítomné požadované funkce. Vymezením funkcí jsou uživatelské příběhy ověřované proti vzniklé aplikaci. V rámci této fáze je důležité ověřit pouze přítomnost a zjevnou funkčnost, zejména pokud je pro ni nutná práce se senzory zařízení nebo komunikace se vzdálenými službami.

Validace použitelnosti

Použitelnost je pro testování relativně komplikovanou disciplínou. Pro správné ověření použitelnosti je potřebná dostatečně velká skupina testujících uživatelů a je dále vhodné velmi cíleně vytvořit testovací scénáře. Dle mého názoru je vhodné vytvářet testovací scénáře nejprve tak, aby měl uživatel dostatek prostoru pro vlastní iniciativu. Tím lze odhalit další potenciální problémy a rozšířit sadu testovacích scénářů. Následně je možné provádět cílenější testování, v rámci kterého se lze zaměřit na uživatelská očekávání v souvislosti s používáním různých prvků. Příkladem při testování online slovníku jsou základní scénáře typu:

- *Zkus vyhledat slovo, které tě zajímá.*
- *Zkus vyhledat fráze a poté si některou vybrat.*

Takto postavené testy pouze ověří schopnost uživatele splnit úkol (testování účinnosti) a mohou odhalit určité chyby. Uživatel například do textového pole vepíše text bez diakritiky. Tím lze následně sestavovat cílenější scénáře, které jsou zaměřené například na očekávání (testování očekávání):

- *Klikni sem. Myslíš, že bude fungovat hledání slova kočka bez diakritiky?*
- *Co myslíš, že se stane, když bys odeslal tento formulář nevyplněný?*

Validace účinnosti

Účinnost lze testovat vhodně vytvořenými testovacími scénáři zaměřenými na úsilí spojené s dokončením úkolů. Úsilí lze obvykle odvodit postavením uživatelských očekávání proti výsledku, který aplikace nabídla doplněná o komentář a celkový pocit uživatele.

- *Očekával jsi <popis očekávání>, nabídla aplikace skutečně to, co jsi očekával?*
- *Jsi celkově spokojen s řešením?*

Lze však testovat i tvrdé metriky a dobu odezvy jednotlivých částí aplikace.

- *Jak rychle si myslíš, že by aplikace měla být schopna zjistit tvou polohu a nabídnout nejbližší nekuřáckou restauraci?*

Kromě času můžeme měřit i přesnost nebo počet kroků, které musí uživatel učinit k naplnění potřeby. Testujeme uživatele, jaké kroky očekává, že bude muset vykonat k dokončení úkolu. Počet kroků se snažíme minimalizovat a velmi dobře v této oblasti může posloužit ambient intelligence. Příkladem budiž uživatel hledající čas příjezdu tramvaje při pravidelném cestování z práce. Jedním extrémem je aplikace vyžadující vložení data, času a názvů výchozí i cílové stanice. Efektivní řešení nabízí aplikace, která doplní výchozí stanici i aktuální čas a nabídne cílovou stanici dle historie. Uživatel po spuštění aplikace pouze použije jediné tlačítko. Aplikace může ale s efektivností jít ještě dále:

- *Čekal bych, že aplikace doplní aktuální čas, zastávku podle GPS a cílovou stanici podle mého předchozího chování. Vyhledávání by mohlo začít nejlépe po tom, co bych zatřásl telefonem.*

Validace bezporuchovosti

Bezporuchovost je ideální stav aplikace, při kterém se aplikace chová zcela dle očekávání a bez vzniku anomálií. Testování bezporuchovosti prostupuje všemi ostatními fázemi validace a měla by být řešena tedy průběžně. V rámci bezporuchovosti zapisujeme podrobný report o vzniklých chybách jako součást prototypu k vyřešení. Po validaci prototypu se opraví vzniklé chyby a probíhá nová validace. V rámci metodiky jsou chyby označovány jako defekty, nicméně standard rozlišuje chyby na třech úrovních dle závažnosti: anomálie, defekt a havárie. Takové rozlišení chyb lze použít u komplexních produktů za účelem nastavení priority při odstraňování většího množství chyb.

Validace udržovatelnosti a přenosnosti

Předpokladem pro udržovatelnost vznikající aplikace je průběžná aktualizace rámcové analýzy vzhledem k nově vytvářeným funkcím. Udržovatelnost lze dále validovat komunikací testerů s vývojářským týmem a zjišťováním

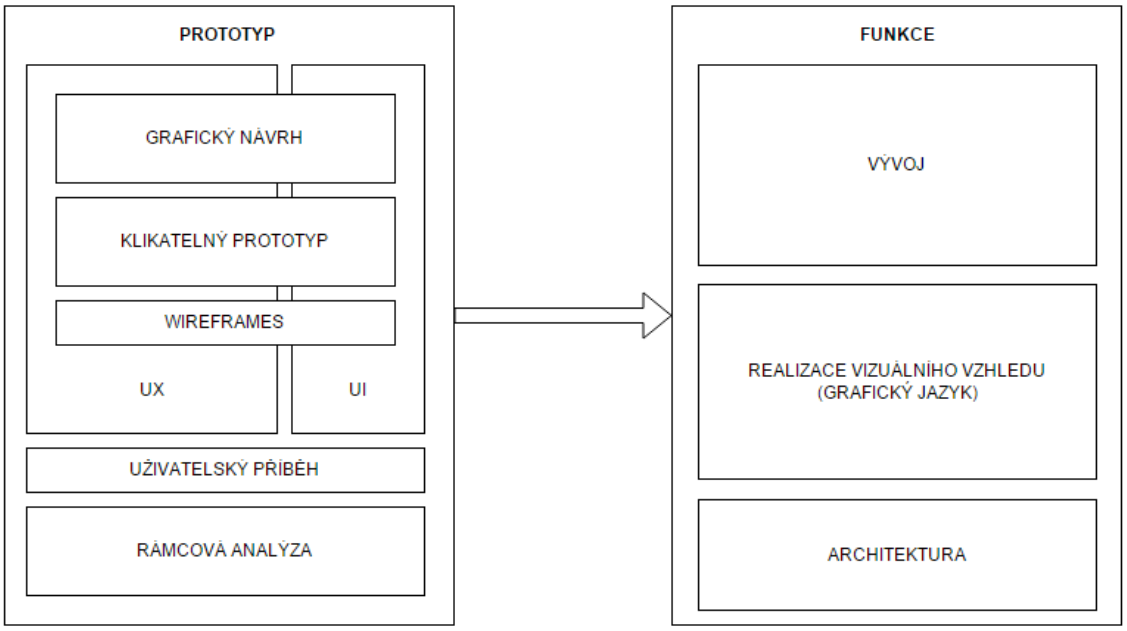
schopnosti realizovat další funkce, související nebo rozšiřující aktuálně validovaný prototyp. Pokud se vrátíme k příkladu s mobilní aplikací, lze vytvořit testování typu:

- *Bude schopen systém v budoucnu ukládat data o vyhledávaných místech?*
- *Bude systém v budoucnu schopen navrhovat i zastávky alternativních dopravců?*
- *Bude systém schopen vyhledat v budoucnu zastávku hlasovým vyhledáváním?*

Samotná přenositelnost lze testovat použitím různých zařízení během procesu uživatelského testování. Stejně jako v případě validace bezporuchovosti lze vytvořit report, v rámci kterého jsou evidovány defekty a nedostatky vzniklé v rámci testování na specifických zařízeních. Vzhledem k vysoké fragmentaci je však nejvhodnější přenosnost testovat samostatně a určit pro tuto fázi jednu odpovědnou osobu.

3.7. Implementace

Z hlediska zdroje dat se aplikace liší tím, zda používají data uložená v paměti zařízení nebo data získaná skrze aplikační rozhraní externích služeb. Z hlediska funkcionality obvykle aplikace nabízí buď data na základě chování uživatele, nebo s pomocí algoritmů odvozuje výstupy na základě různých vstupů. Ve všech případech se aplikace odlišují zvolenou technologií a architektonickým řešením. Nad tímto základem jsou dále stavěny nové funkce a aplikace je rozšiřována. V této technické části práce je proces vývoje rozdělen na návrh architektury, realizaci vzhledu a na vývoj funkcionalit. Východiskem pro technologický proces výroby je podrobně zpracovaný prototyp aplikace vycházející z uživatelského příběhu.



Obr. 19: Popis realizace prototypu ve funkci, zdroj: autor

3.7.1. Architektura

Pro práci s daty a pro definici chování aplikace je používána často vývojářská sada nástrojů, označována jako SDK. Nad SDK je poté stavěna architektura aplikace. Při návrhu architektury je důležité, aby vytvořený kód byl snadno testovatelný, rozšiřitelný a splňoval požadavky definované ve fázích prototypování. Je-li požadavkem UX dostupnost dat v off-line režimu po stisknutí vybraného tlačítka, pak je nutné, aby architektura umožňovala efektivně ukládat data na zvolené úložiště a poskytovat je off-line.

V současné době je při vývoji aplikací používána celá řada návrhových vzorů a doporučení. Za účelem oddělení vizuálního vzhledu a kódu aplikace je vhodné použití návrhového vzoru MVVM. Díky tomuto návrhovému vzoru lze oddělit vývoj grafického vzhledu aplikace od zbytku programového kódu a zároveň s použitím fyzické reprezentace grafického vzhledu, tzv. ViewModelu efektivně provádět testování všech funkcionalit. Toto testování je možné pomocí jednotkových testů a bezesporu se jedná o efektivní způsob, jak zamezit vzniku chyb. Moderní MVVM frameworky využívají princip IoC, s nímž je například možné, aby ViewModely používaly odlišná data v závislosti na tom, zda je aplikace v režimu

návrhu nebo v režimu živého zobrazení. Vývoj také výrazně usnadňuje použití aspektově orientovaného programování, v rámci něhož je aplikace dekorována o aspekty, které je snadné testovat a aplikovat v rámci vybraných entit (obvykle tříd, metod nebo vlastností).

3.7.2. Technická realizace vzhledu

Všechny platformy ve své podstatě nabízejí nějakou formu značkovacího jazyka a umožňují pomocí něj tvorbu vizuálního vzhledu aplikace. V případě aplikací pro OS Windows Phone se standardně pracuje s jazykem XAML založeným na XML, pro iOS je používán Interface Builder a Android definuje vzhled pomocí XML souborů. Všechny platformy nicméně nabízí grafické rozhraní pro tvorbu vizuálního vzhledu s pomocí standardních stavebních prvků platformy. Další inteligentní ovládací prvky lze obvykle integrovat formou různých externích pluginů. V procesu realizace vzhledu je snahou vytvořit věrný a funkční obraz prototypu. Ve fázi tvorby vzhledu může dojít k problémům s naplněním některých UX nebo UI požadavků a je tak zcela regulérní, že proces vývoje je přerušen nebo vyžaduje dopracování a změny v prototypu.

3.7.3. Vývoj

Nad vznikající architekturou a vizuálním vzhledem je možné psát výkonný kód realizující požadavky prototypů a přeměňující je v reálně použitelné funkce. Při vývoji aplikací v týmu je obvykle vhodné použití vybraného nástroje pro správu kódu (tzv. Source Control). Díky správě kódu mohou vývojáři pracovat paralelně na různých funkcích aplikace a zároveň si vzájemně přerozdělovat úkoly v rámci komplikovanějších prototypů. Fáze vývoje tedy spočívá v přeměně prototypu na funkcionalitu postavenou nad aplikační architekturou. Při vývoji lze doporučit aplikaci principů extrémního programování, jehož výhody jsou popsány v kapitole 1.3.1. a dále 4.2.1.

3.8. Testování software

V procesu vývoje software se v současné době používá celá řada různých testů. Ty jsou prováděny od momentu vývoje nové funkce až po doručení nové verze aplikace uživateli. V této části práce jsou popsány typické druhy testů včetně zhodnocení jejich přínosu pro otestování vyvíjené funkcionality. V průběhu textu této kapitoly jsou využívány poznatky ze zdroje (24).

3.8.1. Developer testing

Developer testing je proces, který probíhá neustále při vývoji aplikačního kódu. Vývojář obvykle po napsání jakékoliv funkční části provádí kompilaci a debuguje své řešení. V případě metodiky párového programování se dá tato část nazvat „testem čtyř očí“. Minimálním předpokladem pro splnění této fáze je zkompilovatelný aplikační kód a správné chování aplikace ve vývojářském prostředí. Oprava chyb v této fázi je levná a testování obvykle není časově náročné. Developer testing je vhodný i pro optimalizaci na výkonnost, jelikož umožňuje vývojáři neustále měnit a vyhodnocovat svůj kód. Při vývoji mobilních aplikací je tento způsob testování kriticky důležitý.

3.8.2. Unit testing

O testování jednotek byla zmínka v kapitole 1.3 popisující metodiky XP a TDD. Jednotkou je v případě software metoda třídy. Princip spočívá v napsání testovacího kódu s využitím frameworku pro daný programovací jazyk a následně spuštění (otestování) pomocí tzv. test runneru (spouštěče), který je obvykle přímo integrovaný do nejrozšířenějších IDE.

Metodika XP a TDD doporučuje napsat testy vůči vybrané metodě ještě před samotnou implementací. Díky tomuto postupu je vývojář více zaměřen na vytvoření testů a je tak schopen s předstihem vyřešit scénáře, které by při přímé implementaci mohly uniknout jeho pozornosti. Výhodou jednotkových testů je průběžná kontrola správného chování aplikace a především možnost automatizace procesu spouštění testů pomocí tzv. build serverů. V procesu Continuous Delivery mohou testy

jednotek rapidně snížit riziko vzniku chyby v produkčním prostředí. Unit testy lze doporučit minimálně za účelem pokrytí komplikovanějších modulů ale jejich použití je typické i v případě řešení vzniklých chyb.

3.8.3. Installation testy

Po té, co aplikace opustí vývojářské prostředí, bývají prováděny instalační testy. Ty mají v případě mobilních aplikací (například na rozdíl od webových) bezesporu značný význam, jelikož mobilní aplikace může být doručena na různá zařízení s různými konfiguracemi. Cílem instalačního testu je pouze ověření, zda je možné instalační balík bezchybně nainstalovat na libovolné zařízení dle specifikace. Minimálně kvůli stále rozšiřující se fragmentaci na platformách Android a Windows Phone je vhodné tyto testy provádět. Vzhledem k jejich malé odpovědnosti je však vhodnější začlenit je do testů na dalších úrovních.

3.8.4. Smoke testing

Po úspěšné instalaci aplikace na vybrané prostředí jsou často prováděny smoke testy. Jejich cílem je pozitivní testování aplikace, především za účelem ověření funkčnosti klíčových částí systému. V případě mobilních aplikací se jedná o test spuštění aplikace a přístup do hlavních částí aplikace. V praxi se smoke testy často spojují společně s integračními testy, jelikož při testování komponent v rámci integračního testování je automaticky odhalena nefunkčnost obecnějších částí systému. V případě mobilních aplikací se osobně přikláním k propojení s instalačními testy a integrační testy provést separátně. Důvodem je mimo jiné fakt, že integrační testy obvykle realizuje tým testerů, zatímco smoke testy může provést spolehlivě i jedna odpovědná osoba.

3.8.5. Integrační testy

Standardně se jedná o testy, které v procesu vývoje a distribuce software řídí tým testerů. Cílem je provést kontrolu, zda jednotlivé komponenty systému správně komunikují a zda se systém chová správně na vybrané platformě a zařízení. Jako vhodné řešení se tedy může zdát integrační testování více testery na různých

mobilních zařízeních. Integrační testování bývá často opomíjeno především z toho důvodu, že případné chyby se odhalí i v dalších regulérních fází testování. V případě menších projektů mají pak integrační testy do značné míry tendenci testovat to jisté, jako pozdější testy systémové. Jiným použitím integračních testů může být například zaměření na testování správné komunikace mobilní aplikace s aplikačním rozhraním.

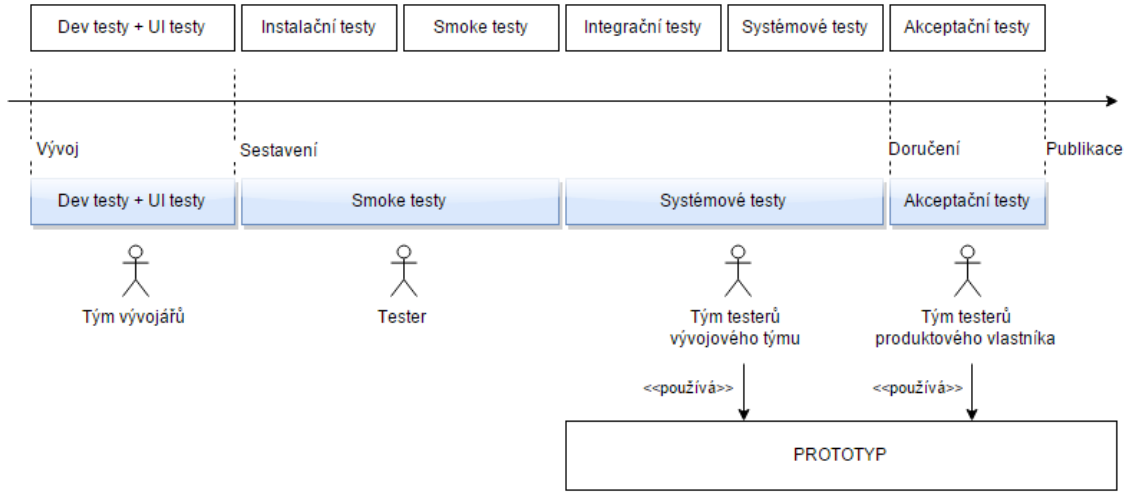
3.8.6. Systémové testy

Systémové testy mají velmi širokou odpovědnost, a pokud se provádí testování aplikací, systémové testy bývají součástí tohoto procesu. Jejich cílem není pouze testování pozitivních scénářů ale i scénářů negativních. Součástí systémových testů v rámci vývoje mobilních aplikací dávají smysl recovery testy pro kontrolu chování aplikace po pádu nebo ukončení v průběhu zpracování. Mezi další testy lze zařadit testování výkonnosti proti definovaným tvrdým metrikám nebo testy související se zabezpečením aplikace. Fáze systémového testování by měla být rozhodně součástí procesu testování celého software. Systémové testy by měly testovat aplikaci na základě připravených scénářů vždy vůči specifikaci vybrané funkce. Touto specifikací je pak především prototyp. Systémové testování bývá obvykle prováděno manuálně odborníky zaměřenými na testování software a v procesu vývoje mobilních aplikací jej lze sloučit s integračními testy, které jsou logicky podmnožinou testů systémových. Kromě toho, právě schopnost integrace je řešena modelem kvality. Pro systémové testování se jeví tedy jako ideální volba využití modelu kvality, popsaného v části 3.6.1. Případně vzniklé anomálie je nutné evidovat a vracet k přepracování do fáze prototypování.

3.8.7. Akceptační testy

Poslední kategorií, která je zmíněna, jsou testy akceptační. Ty jsou běžně prováděny testovacím týmem na straně klienta a jejich cílem je ověření, zda se aplikace chová dle definovaných specifikací. V případě mobilních aplikací lze klientovi doručit instalační balíček, který následně nainstaluje na své zařízení a provede podrobné testy na základě připravených scénářů. Prakticky tedy

akceptační testy obsahují všechny testy předešlé, jen jsou prováděny na straně klienta. V agilním procesu vývoje je doporučena úzká spolupráce zákazníka a vývojářského týmu. Za předpokladu, že v této vazbě panuje dostatečná důvěra a testy popsané v předešlých částí jsou provedeny důsledně, lze akceptační testy neprovádět. Nedůsledné provádění testů a spoléhání na akceptační testování obvykle vede k výraznému prodloužení doby dodávky a to i v procesu Continuous Delivery.



Obr. 20: Náhled na vybrané testy a interakci s prototypem, zdroj: autor

3.9. Publikace

Momentem doručení a akceptace nové verze aplikace může být aplikace publikována cílovým uživatelům skrze virtuální tržiště. Jsou-li splněny veškeré požadavky na kvalitu software, obvykle jsou splněny i požadavky virtuálních tržišť. Existují však určité výjimky, na základě kterých může být aplikace přes svou bezchybnost a kvalitu odmítnuta. Typicky se tak stává v případě použití ochranných známek a značek nebo při použití nevhodného názvu aplikace. Publikací aplikace však život uživatelského příběhu nekončí. Čím více uživatelů aplikaci používá, tím více vzniká riziko objevení neotestovaných případů a zvyšuje se počet hlášených defektů.

3.9.1. Defekty v aplikaci a přístup uživatele

I přes nejsofistikovanější proces testování mobilní aplikace se často do produkční verze aplikace dostane chyba. Dle kapitoly 3.6.1 by součástí aplikace měl být mechanismus pro odhalení a logování chyb. Některé mobilní aplikace umožňují zasílat zpětnou vazbu, jiné shromažďují vnitřně informace o výjimkách a ty jsou příležitostně na pozadí odesílány vývojářům. Uživatelé chyby nacházejí prakticky neustále v důsledku jejich nepředvídatelného chování nebo odlišných očekávání. Chyby v produkční verzi aplikace je nutné shromažďovat, získávat o nich maximum informací a předávat je k nápravě. Uživatelé jsou na chyby zvyklí a často je v menší či větší míře očekávají a tolerují. Především chyby, které souvisí s bezpečností, by však měly být co nejrychleji odstraněny.

„Slabiny vznikají vinou spěchu a snahy být rychlejší než konkurence, z důvodu používání zastaralých programátorských postupů, případně i z jiných důvodů. Podstatné je, jak dlouho chyby v programu existují. Přesněji řečeno, důležité je, jak dlouhá doba uplynula od uvedení programu s konkrétní slabinou na trh do zveřejnění slabiny, případně do vydání opravy.“ (1)

4. Použití metodiky v procesu vývoje

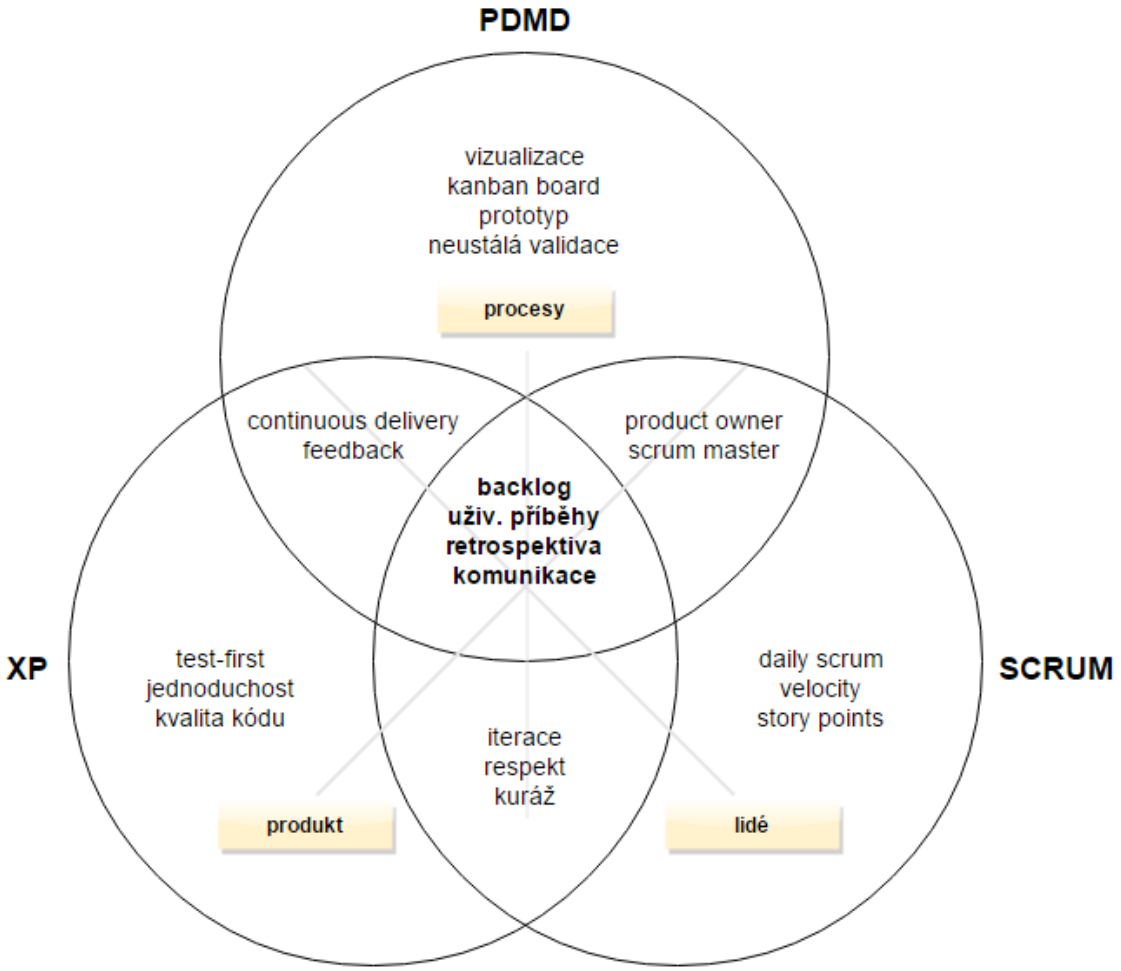
Metodika PDMD se zaměřuje především na fázi technologické výroby aplikace a navrhuje základní možnosti, jakými může být realizována. PDMD je soběstačná metodika, přejímá nejlepší praktiky z již nejrozšířenějších agilních metodik, mezi které patří například Extrémní programování nebo SCRUM ale zároveň umožňuje přímé propojení s těmito metodikami, čímž se značně rozšiřují možnosti jejího využití. V této kapitole jsou naznačeny možnosti propojení PDMD s dalšími metodikami a zdůrazněna je přidaná hodnota PDMD vůči ostatním metodikám. Cílem této kapitoly je prokázat schopnost aplikace metodiky do skutečného procesu výroby aplikace.

4.1. Samostatné použití metodiky

Metodika PDMD vychází z několika artefaktů, které jsou společné dalším agilním metodikám a patří mezi ověřené praktiky při vývoji software. Mezi tyto praktiky můžeme zařadit tvorbu backlogu obsahujícího uživatelské příběhy a realizaci těchto příběhů ve funkce v iterativním procesu. Z metodiky XP je převzat princip Continuous Delivery zajišťující neustálé doručování nových funkcí uživatelům. Z metodiky SCRUM, která je především zaměřena na projektový management a spolupráci lidí jsou využity principy související s komunikací mezi produktovým vlastníkem a projektovým manažerem. Metodika tak bezesporu pokrývá oblast procesu od požadavků až po realizaci uživatelského příběhu ve funkci.

Metodika PDMD dále používá praktiky z metody Kanbanu, mezi které patří především vizualizace vývoje nových funkcí na Kanban board. Zároveň využívá schopnost Kanbanu podporujícího efektivnější výrobu limitací počtu uživatelských příběhů v jednotlivých fázích výroby. Díky tomu jsou uživatelské příběhy spolehlivěji odbavovány a je zamezeno přehlcení týmu nedořešenými úkoly. Největší přidanou hodnotou a rozdílem oproti ostatním metodikám je snaha o vytvoření prototypu v úvodní fázi procesu vývoje nové funkce. Prototyp je v PDMD nejen podkladem pro práci v dalších fázích ale zároveň se stává specifikací, díky které je možné předejít nadbytečným nákladům. V neposlední řadě je prototyp

nástrojem pro komunikaci mezi dodavatelem a zákazníkem a v každé fázi procesu výroby nové funkce je stěžejním artefaktem metodiky, umožňujícím neustálou validaci. Zasazení a odlišnosti metodiky jsou zobrazeny na Obr. 21.



Obr. 21: Metodiky a jejich hlavní hodnoty, zdroj: autor

Z obrázku je patrné, že PDMD má potenciál fungovat soběstačně ale zároveň umožňuje využít další praktiky z XP pro zlepšení kvality kódu nebo praktiky z metodiky SCRUM díky nimž lze zpřesnit způsob komunikace uvnitř týmu a lépe provádět projektový management. Metodika PDMD je agilní metodika, která je použitelná ve všech případech vývoje mobilní aplikace, kdy zákazník svěří vývojářskému týmu svou důvěru. S tím souvisí především fakt, že metodika nemá

předpoklad pro použití, resp. nemůže být efektivní tam, kde před zahájením vývoje existují konkrétní specifikace vyvíjených funkcionalit (například prototypy).

4.2. Možnosti propojení s dalšími metodikami

Součástí metodiky PDMD je již velká část myšlenek Kanbanu. Kanban přitom není sám o sobě metodikou ale spíše technologickým vylepšením stávajících procesů. Mezi základní praktiky, které vylepšují navrženou metodiku pro vývoj mobilních aplikací, patří vizualizace. Ať už je vytvářena jakákoliv funkce, její aktuální stav je vizuálně vidět na Kanban boardu. Právě Kanban board je jedním z principů, které pomáhají vývojářským týmům více kooperovat a mít vzájemně přehled o veškeré práci a stavu produktu, čímž se metodika PDMD velmi přibližuje ke SCRUMu i XP.

Podstatný rozdíl a filosofická podstata metodiky PDMD spočívá v zakomponování fáze prototypování dovnitř procesu Continuous Delivery. Proces jako takový však nenarušuje interakci ani s metodikou SCRUM ani s XP. Proces neustálého doručování nových funkce klade vzhledem k podstatě mobilních aplikací velký důraz na správnost. PDMD zavádí praktiku striktní validace dvou klíčových artefaktů, prototypu a funkce v podobě nové aplikace, přičemž proces validace používá jako dva základní prostředky zmíněný prototyp a vybraný model kvality. Tento proces, prototypem řízená validace, výrazně snižuje riziko vzniku chyb a především i eliminuje vznik zbytečných nákladů na vývoj zbytečných nebo nevhodně implementovaných funkcí.

Metodika PDMD maximálně zjednodušuje systém rolí a definuje pouze zákazníka (Product Owner) a dodavatele (Project manager / SCRUM Master / Team leader). Zda je na straně projektového manažera pouze jedna osoba nebo celý tým, není klíčové. I na základě této minimalizace lze provést další rozšíření PDMD a propojení s jinými metodikami. V dalších částech této kapitoly jsou popsány možnosti propojení PDMD na dvou oblíbených metodikách: XP a Scrum.

4.2.1. PDMD + XP

Propojení metodiky PDMD a XP umožňuje rozšířit proces vývoje mobilní aplikace o řadu užitečných praktik a principů. V rámci propojení těchto metodik je proces Continuous Delivery doplněn o XP iterace, v rámci kterých probíhá plánování a retrospektiva. Metodika XP přináší řadu užitečných praktik v souvislosti s procesem testování na úrovni programového kódu. Zatímco PDMD pouze vyžaduje testování fáze vývoje po dokončení implementace, metodika XP vyžaduje vznik jednotkového testu a teprve následnou implementaci funkce.

Další užitečné praktiky souvisí s kvalitou zdrojového kódu. Kvalitní zdrojový kód znamená snadnou správu a rychlé a levné rozšiřování. Především z těchto důvodů je snahou XP zavádět praktiky, jako např.: párové programování, sdílení společných pracovních prostor nebo myšlenku společného vlastnictví. Na kvalitu kódu má ale vliv i neustálý proces refaktoringu, redukce opakování a snaha o maximálně jednoduchý vývoj pouze nezbytných funkcí. Vhodné je poznamenat, že žádná z těchto praktik není v rozporu s návrhem PDMD a tyto praktiky a principy lze doporučit.

Propojení metodik PDMD a XP ve své podstatě přináší řadu zlepšení v oblasti kvality kódu ve fázi vývoje, bez toho aniž by byl jakkoliv ovlivněn základní rámec PDMD. Kromě toho, myšlenka prototypování přináší větší potenciál pro vznik spolehlivých jednotkových testů, vůči kterým je následně dopsána požadovaná funkcionalita.

4.2.2. PDMD + SCRUM

SCRUM je metodika, která věnuje mnoho úsilí k interakci mezi rolemi uvnitř procesu. Stejně jako PDMD definuje i SCRUM role produktového vlastníka a projektového manažera v podobě SCRUM mastera. Z hlediska PDMD SCRUM inklinuje k neformální spolupráci a ke komunikaci se zákazníkem často zapojuje celý tým. Všechny tyto principy lze jednoduše zařadit do procesu vývoje aplikací pomocí PDMD. PDMD kromě toho vkládá do procesu dva velmi důležité artefakty. Tím prvním je neustále aktualizovaná rámcová analýza a dále aktuální rámcová

dokumentace. Tyto jednoduché dokumenty umožňují získat snadno komplexní představu o produktu, jeho stavu a rozsahu a vizualizují pohled na realizovanou aplikaci.

Stejně jako v případě XP i SCRUM vyžaduje denní schůzky členů týmu, během kterých lze odstraňovat překážky při vývoji aplikace. Na konci iterací, tzv. sprintů nakonec probíhá retrospektiva, kterou doporučuje i metodika PDMD a v rámci které lze provést mimo jiné i aktualizaci rámcové analýzy. Rozdíl oproti metodice SCRUM spočívá především ve vizualizaci všech řešených uživatelských příběhů na Kanban boardu a omezení počtu řešených uživatelských příběhů v rámci každé fáze. To však nečiní žádný problém, protože tento proces pouze odstraňuje riziko vzniku nedořešených úkolů v jednotlivých fázích procesu výroby. Uživatelské příběhy pak nejsou odebírány z Backlogu průběžně ale jsou na začátku sprintu naplánovány do fáze „K vyřešení“. Nástroje ke správnému odhadu kapacity sprintu jsou Planning poker a především pak komunikace SCRUM mastera s produktovým vlastníkem a zásah do Backlogu v případě potřeby.

4.2.3. PDMD + SCRUM/XP

Vzhledem k tomu, že metodika XP přináší řadu efektivních praktik v oblasti vývoje aplikací a zlepšuje kvalitu zdrojového kódu, je často propojována s metodikou SCRUM, která je zaměřená především na praktiky zlepšující kooperaci lidí a projektový management. Připojení metodiky PDMD mezi SCRUM a XP má potenciál přinést synergii v podobě zlepšení procesů, kvality kódu ale i důsledné validace a odstranění zbytečných nákladů na základě působení a použití prototypu při technologickém procesu vývoje mobilních aplikací. Z Obr. 21 je patrné, že všechny tři metodiky mají společný základ a dále každá s každou mají určité společné praktiky, které lze snadno rozšířit o nové specifické možnosti související vždy s konkrétní vybranou metodikou

5. Shrnutí výsledků

Diplomová práce v první části analyzovala současné metodiky, používané pro vývoj software a nastínila základní rozdíly mezi tradičními a agilními metodikami. V rámci této analýzy bylo poukázáno na přednosti i nedostatky jednotlivých metodik a dále byly identifikovány praktiky a procesy při vývoji software. Dále bylo poukázáno na problém, že ačkoliv agilní metodiky bezesporu přináší větší úspěšnost projektů, vývojářské týmy je mají často problém důsledně aplikovat. Diplomová práce poukázala také na fakt, že žádná z vybraných metodik není přímo zaměřená na technologický proces tvorby mobilní aplikace, ačkoliv teoreticky aplikovatelné jsou všechny. Identifikace procesů a nalezení klíčových odlišností při vývoji mobilních aplikací bylo dále zkoumáno v kapitole 2. Z této kapitoly byl odvozen závěr, že jednou z klíčových fází při tvorbě mobilních aplikací je její vizuální vzhled a s tím související použitelnost aplikace.

Při návrhu univerzální metodiky pro vývoj mobilních aplikací bylo položeno několik předpokladů na základě poznatků z předešlých kapitol a dále byl vytvořen funkční rámec vycházející ze současných agilních metodik. Metodika dále tento proces zlepšila vznikem fáze prototypování a dále neustálou validací všech výrobních fází za účelem zamezení vzniku chyb a nadbytečných nákladů. Jako dvě hlavní role v procesu vývoje byl definován produktový vlastník a projektový manažer zastupující dodavatele ve formě formální nebo neformální spolupráce. Mezi klíčové prvky metodiky bylo zahrnuto neustálé doručování nových funkcí od produktového vlastníka až k uživateli. Metodika dále navrhla oproti běžným agilním metodikám začlenění rámcové analýzy do procesu vzniku aplikace jako představu o produktu ze strany produktového vlastníka a rámcovou dokumentaci jako základní retrospektivní pohled na vytvářený systém z pohledu dodavatele. Za účelem zlepšení procesu metodika přijala metody vizualizace a nastavení maximální kapacity fází vývoje z metody Kanban.

Klíčový artefakt metodiky, prototyp, byl popsán v kapitole 3.5 včetně jeho součástí a oblastí, které do této oblasti zasahují až po jeho realizaci. Velká část práce byla následně věnována fázím validace tohoto prototypu a následně vývoji

zakončenému testováním s využitím zmíněného prototypu. Prototyp metodika nadefinovala nejen jako fázi procesu vývoje mobilní aplikace i jako prvek, který umožňuje neustálou kontrolu nad správnou funkčností řešení s využitím zvoleného modelu kvality. Tímto modelem byl v metodice zvolen standard ISO 9126 pro svou jednoduchost a schopnost pokrytí požadavků na kvalitu mobilních aplikací.

Přes svou soběstačnost bylo v závěru poukázáno na fakt, že vzhledem k filosofii metodiky, jejímž základem je fáze prototypování, není metodika použitelná na vývoj zcela všech mobilních aplikací se stejnou efektivitou. Metodika dále neposlouží ke svému účelu v případech, kdy již existují striktní požadavky na vytvářenou aplikaci ve formě specifikací a již hotových prototypů. V poslední části diplomové práce bylo poukázáno nejen na schopnost metodiky fungovat zcela nezávisle, ale především bylo nastíněno její rozšíření technickým směrem nebo směrem zpřesnění požadavků na management s využitím metodiky SCRUM.

6. Závěry a doporučení

Diplomová práce naplnila cíl nalezením metodiky vývoje mobilních aplikací s názvem *„Prototypem řízený vývoj mobilních aplikací“*. Navržená metodika vychází ze současných agilních metodik a obohacuje je o vstupní analýzu a výstupní rámcovou dokumentaci za účelem pohledu na celý vytvářený systém. Důležitým prvkem této metodiky je prototyp, který definuje požadavky na vytvářené funkce a následně slouží jako objekt usnadňující komunikaci mezi zadavatelem a dodavatelem. Zároveň je prototyp používán k neustálé validaci vytvářené funkce proti zvolenému modelu kvality. Tento klíčový proces metodiky je nazván *„Prototypem řízená validace“* a probíhá v průběhu celého vývoje. V neposlední řadě metodika využívá k vizualizaci procesu tzv. Kanban Board a myšlenku metody Kanban související s omezením kapacity jednotlivých výrobních fází.

Metodika sdílí stejné obecné a léty praxí ověřené principy jako jiné agilní metodiky a nepochybně umožňuje usnadnit vývoj širokého spektra mobilních aplikací s výjimkou těch, jejichž specifikace nebo prototyp již vznikl mimo realizační proces metodiky. Přínosem metodiky je především její přizpůsobení požadavkům na vývoj úspěšných mobilních aplikací a to se všemi prostředky, které byly uvedeny v předešlém odstavci. Přestože navržená metodika je zcela nezávislá a z hlediska výrobního procesu je velmi striktní, oblasti kvality kódu a projektového managementu navrhuje s dostatečnou abstrakcí, aby mohla být snadno sloučena s dalšími metodikami, jako například extrémní programování nebo SCRUM.

Seznam použité literatury

1. **Sager, Ira.** Before IPhone and Android Came Simon, the First Smartphone. *Bloomberg Business.* [Online] 29. 6 2012. [Citace: 31. 1 2015.] http://www.bloomberg.com/bw/articles/2012-06-29/before-iphone-and-android-came-simon-the-first-smartphone.

2. **Royce, Dr. Winston W.** MANAGING THE DEVELOPMENT OF LARGE SOFTWARE SYSTEMS. *Computer Science University of Maryland.* [Online] [Citace: 28. 1 2015.] http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf.

3. *Software Development: Agile vs. Traditional.* **Stoica, Marian, Mircea, Marinela a Ghilic-Micu, Bogdan.** 17, 2013. 14531305.

4. **Popelka, Vladimír.** *Srovnávací analýza metodik vývoje.* Praha : Vysoká škola ekonomická v Praze, 2009.

5. **Beck, Kent, a další.** Manifesto for Agile Software Development. *Manifesto for Agile Software Development.* [Online] 2001. [Citace: 13. 1 2015.] http://www.agilemanifesto.org.

6. **Beck, Kent.** *Extreme Programming Explained: Embrace Change, 2nd Edition (The XP Series).* místo neznámé : Addison-Wesley, 2004. 978-0321278654.

7. **Schwaber, Ken.** *Agile Project Management with Scrum (Developer Best Practices).* Redmond : Microsoft Press, 2014. 978-0735619937.

8. **Hajdin, Bc. Tomáš.** *Agilní metodiky vývoje software, Diplomová práce.* Brno : Masarykova univerzita v Brně, Fakulta informatiky, 2005.

9. **Humble, Jez.** *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation.* Upper Saddle River, NJ : Addison-Wesley Professional, 2010. 860-1401501176.

10. **Fowler, Martin.** The New Methodology. *Martin Fowler.* [Online] 13. 12 2015. [Citace: 20. 1 2015.] http://www.martinfowler.com/articles/newMethodology.html.

11. **Cohn, Mike.** Agile Succeeds Three Times More Often Than Waterfall. *Mountain Goat Software.* [Online] 13. 2 2012. [Citace: 16. 1 2015.] Agile Succeeds Three Times More Often Than Waterfall.

12. **Devi, Vijaya.** Scrum Alliance. *Traditional and Agile Methods: An Interpretation.* [Online] 23. 1 2013. [Citace: 23. 1 2015.] https://www.scrumalliance.org/community/articles/2013/january/traditional-and-agile-methods-an-interpretation.

13. **Fetto, John.** Americans spend 58 minutes a day on their smartphones. *Experian Marketing Services.* [Online] 28. 5 2013. http://www.experian.com/blogs/marketing-forward/2013/05/28/americans-spend-58-minutes-a-day-on-their-smartphones/?WT.srch=PR_EMS_smartphones_052813_press.

14. **Weinberger, Matt.** Design is more important than technology. *CITEWorld.* [Online] 13. 6 2014. [Citace: 2. 3 2015.] http://www.citeworld.com/article/2363200/development/design-is-more-important-than-technology.html.

15. **Johnson, Bobbie.** Google reveals mobile plans. *The Gurdian.* [Online] 5. 11 2007. [Citace: 31. 1 2015.] http://www.theguardian.com/technology/2007/nov/05/google.mobilephones.

16. **Statista, Inc.** Number of apps available in leading app stores as of July 2014. *The Statistics Portal.* [Online] 30. 7 2014. [Citace: 31. 1 2015.] http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/.

17. **Sweney, Mark.** Apple launches iPhone. *The Guardian.* [Online] 9. 1 2007. [Citace: 31. 1 2015.] http://www.theguardian.com/technology/2007/jan/09/news.newmedia.

18. **Microsoft, s.r.o.** Microsoft Unveils Windows Phone 7 Series. *Microsoft News Center.* [Online] 15. 2 2010. [Citace: 30. 1 2015.] http://news.microsoft.com/2010/02/15/microsoft-unveils-windows-phone-7-series.

19. **Fowler, Martin.** UserStory. [Online] 22. 4 2013. [Citace: 26. 2 2015.] http://martinfowler.com/bliki/UserStory.html.

20. **Warfel, Todd Zaki.** *Prototyping: A Practitioner's Guide.* New York : Rosenfeld Media, 2009. 978-1933820217.

21. **http://www.usability.gov.** Prototyping. [Online] [Citace: 26. 2 2015.] http://www.usability.gov/how-to-and-tools/methods/prototyping.html.

22. **Singh, Inderpal.** Different Software Quality Model. [Online] 5 2013. [Citace: 25. 2 2015.] http://www.academia.edu/4899160/Different_Software_Quality_Model.

23. **Nielsen, Jakob.** Why You Only Need to Test with 5 Users. *Nielsen Norman Group.* [Online] 19. 3 2000. [Citace: 2. 3 2015.] http://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/.

24. **Patton, Ron.** *Testování softwaru. Vyd. 1.* Praha : Computer Press, 2002. ISBN 80-7226-636-5.

25. **Nápravník, Jiří.** Slabiny Windows 8. Proč se o ně mají zajímat běžní uživatelé? *Hospodářské noviny.* [Online] 28. 11 2012. [Citace: 15. 2 2015.] http://napravnik.blog.ihned.cz/c1-58845300-slabiny-windows-8-proc-se-o-ne-maji-zajimat-bezni-uzivatele.

26. **Apple.** App Store Rings in 2015 with New Records. *Apple Press Info.* [Online] 8. 1 2015. http://www.apple.com/pr/library/2015/01/08App-Store-Rings-in-2015-with-New-Records.html.

27. ***ISO/IEC 9126-1:2001, Software engineering — Product quality.* 1991.**

28. **Pergl, Ing. Robert.** *Metody řízení softwarových projektů využívající moderní paradigmata (Disertační práce).* Praha : Česká zemědělská univerzita v Praze, 2008.

Seznam tabulek

Seznam obrázků

FIM UHK
UNIVERZITA HRADEC KRÁLOVÉ
Fakulta informatiky a managementu
Rokitanského 62, 500 03 Hradec Králové, tel.: 493 331 111, fax: 493 332 235

Zadání k závěrečné práci

Jméno a příjmení studenta: **Miroslav Holec**

Obor studia: Aplikovaná informatika (2)
Jméno a příjmení vedoucího práce: **Filip Malý**

Název práce:
Metodika vývoje mobilních aplikací

Název práce v AJ:
Methodology of mobile applications development

Podtitul práce:

Podtitul práce v AJ:

Cíl práce: Cílem práce je identifikovat procesy, které jsou součástí tvorby mobilní aplikace a najít efektivní univerzální metodiku, na jejíž základě je možné mobilní aplikace vytvářet.

Osnova práce:
1. Úvod
2. Cíle a metodika zpracování
3. Identifikace procesů a fází tvorby aplikace
4. Agilní metodiky a jejich využití
5. Proces vývoje aplikace
6. Shrnutí výsledků
7. Závěry a doporučení
8. Literatura

Projednáno dne: 9. 10. 2014

Podpis studenta

Podpis vedoucího práce