KAPITOLA 3

Průběh projektu

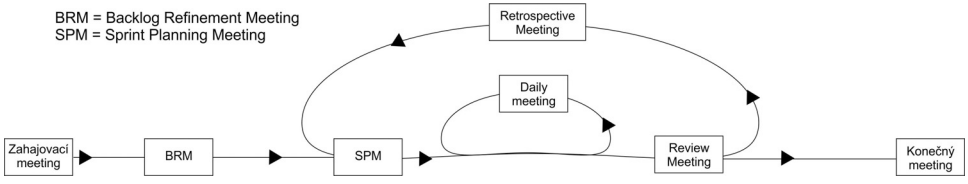
V této kapitole:

- Základní pojmy a artefakty
- Meetingy
- Měření v projektu

Již jsme popsali mnoho stránek, abychom si řekli, jak se vlastně software vyvíjí, co potřebujeme k vývoji softwaru, jak má obecně vývoj vypadat. Již jsme si také pověděli něco o rolích, ve kterých mohou lidé v rámci SCRUMu vystupovat. Nyní je tedy konečně možné přistoupit k samotnému vývoji v rámci metodiky SCRUM. Metodika SCRUM přitom definuje několik zásadních činností (zaměřených především na komunikaci) a tyto se neustále opakují. Nejprve se tedy podíváme, jak vypadá vývoj pomocí metodiky SCRUM.

SCRUM principálně není složitý. A není se čemu divit. Cílem přece je, aby se jednalo o relativně jednoduchou metodiku.

Upozornění: Pozor – jednoduchá metodika neznamená, že je jednoduché ji správně použít. Jednoduchá je v tom smyslu, že neobsahuje desítky rolí, desítky dokumentů, desítky nařízení a předpisů, které vás spoutávají. Na druhou stranu – existence přesných předpisů a procesů může mnoha lidem vyhovovat – ví přesně, co mají kdy dělat. To ve SCRUMu není, tam musíte z velké části sami pochopit, co přesně se od vás chce. Uvidíte, že pro pochopení principů SCRUMu je třeba pochopit poměrně malé množství pojmů. Samotný průběh SCRUMu také není na popis složitý. V tom je jeho jednoduchost.



Na obrázku vidíte průběh celého vývojového cyklu řízeného metodikou SCRUM. Opravdu strukturálně poměrně jednoduché, nemyslíte? Průběh vývojového cyklu skutečně není nijak složitý. V podstatě můžeme celý cyklus rozdělit do tří etap:

1. **Zahájení** – fáze projektu do počátku samotného vývoje, cílem je ujasnění formy spolupráce, definice toho, co se vlastně bude vyvíjet, rozdělení rolí a kompetencí, vzájemné seznámení vývojového týmu.

2. **Samotný vývoj** – fáze, ve které probíhá samotný vývoj aplikace. Tento vývoj je, jak už jsme si několikrát naznačili, iterativní. Jednotlivé iterace se nazývají sprinty – budeme se jim věnovat hned v následující kapitole.
3. **Ukončení** – fáze projektu, ve které je výsledný produkt finálně otestován, akceptován, nasazen a kdy je projekt oficiálně ukončen.

Upozornění: Toto je pochopitelně členění projektu, který úspěšně ukončíme a řádně předáme. V knize pochopitelně řešíme i problémy, a dokonce i stavy, které vedou k ukončení projektu, nicméně hlavní proud knihy pochopitelně popisuje situaci, kdy projekt vedeme řádně metodikou SCRUM a kdy také projekt řádně ukončíme.

Jakmile proběhne fáze **Zahájení**, vstupujeme do fáze **Vývoje**. V této fázi pak setrváváme až do závěru projektu. V této fázi probíhají jednotlivé sprinty, které mají formálně vždy stejný průběh. Slovo formálně je velmi důležité, protože reálně se sprinty samozřejmě liší, co se samotné práce týká. Ale velká část aspektů sprintu je stále stejná – zahájení, plánování, kontrola, měření atd. Ve skutečnosti přesně tohle je to, co očekáváme od metodiky. Jistotu a stabilitu v procesu, nikoliv v samotném obsahu práce. Tento obsah práce je pochopitelně určen tím, co vlastně chceme vyvinout.

Pojďme se nyní nejprve podívat na základní pojmy a artefakty, se kterými SCRUM pracuje. Uvidíte, že jich není mnoho a nejsou ani nijak obtížné na vysvětlení. Opět – tím není řečeno, že práce s nimi je triviální.

Tip: Co ovšem platí stoprocentně, je fakt, že na pochopení principů je metodika SCRUM rozhodně příjemnější než mnohé daleko složitější metodiky. Zdůrazňuji – pochopení základních principů. Ale pochopení základního principu je základem toho, abychom vůbec mohli začít. Ve SCRUMu je možné tyto základní principy popsat během deseti minut. A to vůbec není malá výhoda…

Základní pojmy a artefakty

SCRUM, stejně jako jakákoliv metodika, používá některé své pojmy a artefakty, se kterými dále pracuje. Na rozdíl od mnoha jiných metodik jich není naštěstí mnoho, ale o to je důležitější těch několik základních dobře pochopit.

User Story

Dostáváme se k prvnímu pojmu. A to pojmu velmi důležitému. Začneme lehce zeširoka. Jestliže máme vyvinout jakýkoliv software, první, co potřebujeme, je znát požadavky svého zákazníka. Tohle pochopitelně platí bez ohledu na metodiku. Nemůžeme vytvo-

řit software, pokud nevíme, jaký software máme tvořit. Mimochodem, nutnost znalosti požadavků zákazníka pouze pro IT projekty, tohle je obecné.

Stojíme však před problémem, jak vlastně s požadavky zacházet. Správa požadavků je samostatná vědní disciplína, o které bylo napsáno již několik kvalitních publikací. Jenže – my při správě požadavků máme dvě protichůdné potřeby:

- Pochopitelnost a jednoznačnost pro vývojáře
- Pochopitelnost a jednoduchost pro zákazníka

Poznámka: Proč jsou vlastně protichůdné? Základním důvodem je, že vývojář a zákazník nehovoří stejnou řečí. Zákazník (firma zákazníka – právní zákazník) má určitý obor činnosti – z toho logicky vyplývá, že jednotliví lidé rozumí právě tomuto oboru. Nikoliv informačním technologiím. Z toho vyplývá, že řečí tohoto oboru budou chtít komunikovat. Oni myslí například ve fakturách, objednávkách, projektech. Oproti tomu vývojář bude chtít komunikovat ve své řeči – tedy v databázích, objektech a podobných. A to je rozpor. Jedna strana rozumí jednomu a nerozumí druhému, druhá strana totéž, jen v obráceném gardu. Logické je, že přizpůsobit by se měli zejména vývojáři. To proto, že ve výsledku bude software používán zákazníkem. Proto požadavky zapsané tak, aby to vyhovovalo zákazníkovi, budou určitě problematické pro vývojáře a naopak.

Zákazníka ve skutečnosti nějaké databáze a další „ajťácké věci" nezajímají. Proč by měly? Počítače a další informační technologie jsou pro něj nikoliv cílem, ale nástrojem. Zamysleme se nad tím, čeho chceme ve SCRUMu vlastně dosáhnout. Stále téhož – jednoduchosti, přátelského prozákaznického přístupu, funkčního softwaru, omezení byrokracie. A to je podstatou právě User stories.

Co vlastně je User story (uživatelský příběh)? Zkuste si chvíli představit, že jste nikoliv vývojář, ale zákazník či uživatel. To vlastně není ani tak těžké, protože pro mnoho programů opravdu zákazníky či uživateli opravdu jste.

Přemýšlíte v té době nad implementací? Nikoliv. Přemýšlíte v té době v pojmech oboru informačních technologií? Rovněž nikoliv. Ve skutečnosti je vám úplně jedno, jak systém funguje uvnitř (kromě odborného zájmu). Dokonce si můžete říci – i kdyby v počítači běhal trpaslík a prováděl jednotlivé operace, důležitý pro uživatele je výsledek. Jste-li uživatelem, pak přemýšlíte v pojmech úkolů, které máte pomocí daného softwaru splnit. Takže například takto:

- Stisknutím určeného tlačítka chci změnit velikost písma
- Stisknutím určitého tlačítka chci vytisknout zvolenou fakturu
- Vybráním ze seznamu chci určit typ zákazníka
- …

Takto bychom mohli pokračovat. A právě takový přístup je základem (pouze základem) principu User stories (uživatelských příběhů). Zjednodušeně nám jde o to, že pomocí

jednoduchých vět definujeme jednotlivé scénáře (úkoly), které uživatel v určité roli (v roli uživatele výsledného softwaru, nikoliv při vývoji softwaru) bude se systémem provádět.

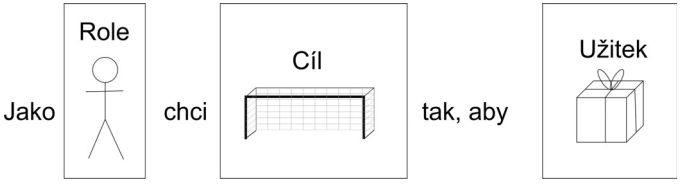
Upozornění: Celý princip uživatelských příběhů tedy využívá pohled zákazníka. Kromě toho, že je to pro zákazníka výhodné, protože on vidí od počátku definici systému ze svého pohledu, ve své řeči, ve svých pojmech, bez složitostí oboru informačních technologií, je to v jednom důležitém aspektu důležité a užitečné i pro vývojáře. Nutí ho to totiž myslet jako zákazník, alespoň v určitých aspektech. To je také jeden z cílu SCRUMu – aby všichni chápali počítač jako nástroj. Vývojář se nemá příliš zabývat tím, aby projekt byl krásný, ale tím, aby byl především funkční. Aby program umožnil splnit úkol, který uživatel má zadaný a který splnit potřebuje. A to tak, jak si zákazník přeje. Uživatelské scénáře založené čistě na uživatelském popisu jednotlivých funkcionalit tomu určitě vhodně napomáhají.

Tedy User story je uživatelský popis toho, co by systém měl dělat. Nikoliv toho, jak by to měl dělat. To uživatele, jak bylo výše zmíněno, vůbec nezajímá. Jednotlivé funkcionality systému tedy budou ztotožněny s jednotlivými úkoly jednotlivých rolí tak, jak je chápe zákazník. Přestože se přitom chceme vyhnout byrokracii, budeme chtít, aby tyto popisy nebyly zcela živelné, ale aby byly v určitém rozumném formátu. Nemusíte se ovšem bát, formát bude snadný – jde prostě o to, aby každý uživatelský příběh nevypadal jinak a aby vývojáři nemuseli studovat složité epické popisy.

Tip: Máte-li spoustu dat, která jsou strukturálně totožná, pak vždy budeme upřednostňovat stav, kdy opravdu budou stejná také vizuálně. Uveďme příklad – kartotékové lístky. Strukturálně stejná data, která jsme sjednotili také vizuálně. Laicky a lidsky – každá kartička vypadá stejně jako všechny ostatní. A když vezmeme do ruky kartičku, tak okamžitě víme, kde máme hledat jednotlivé informace. O nic víc nejde. Samozřejmě, můžeme zbytečně byrokratizovat, vymýšlet složité formuláře. Nebo nic vymýšlet nemusíme, můžeme na to jít jednoduše. Jen prostě zachováme určitou strukturu.

Poznámka: Možná vás napadne, pokud znáte jiné metodiky, že uživatelské příběhy jsou poměrně podobné uživatelským případům (use cases). Ano, opravdu jsou zde určité podobnosti. Obě metody popisují situaci z pohledu uživatele, obě metody popisují úkoly, které má uživatel se systémem provádět. Ale jsou zde také rozdíly. Uživatelské příběhy SCRUMu pouze popisují daný úkol, zatímco uživatelské případy rozepisují posloupnost interakcí se systémem. Celkově je metoda uživatelských případů formálnější. V mnoha projektech se uživatelské příběhy zapisují na malé kartičky, zatímco při používání uživatelských případů se tvoří běžný písemný dokument. Uživatelské příběhy jsou také méně obsáhlé a popisují menší úkoly, zatímco uživatelské případy popisují často celou funkcionalitu.

A proto i uživatelské příběhy mají určitou jednotnou podobu. Ta je znázorněna na obrázku.



User story (neboli uživatelský příběh, jedná se o jeden a tentýž pojem, používám oba, protože oba se v praxi opravdu objevují) má tedy tři základní části:

1. **Definice role** – v každé roli, ve které se jako uživatel nově vyvíjeného systému můžete nacházet, máte určité úkoly – z těch vyplývají jednotlivé uživatelské příběhy. Přitom je velmi důležité, aby tyto scénáře byly spárovány se svými rolemi. Role a příběhy, jak už jsme si řekli v kapitole o rolích, jsou svázány. Příběh má smysl pouze tehdy, je-li zároveň řečeno, kdo tento příběh má prožívat. Proto nejprve definujeme roli, které se daný příběh týká.

Tip: Role opět chápeme tak, jak nám je definuje zákazník. Žádné složitosti. Proč taky? Ptejme se zákazníka – v jakých rolích může uživatel vystupovat. Ve skutečnosti platí zlatá zásada – rolí jen tolik, kolik je nezbytně nutné. Role pojmenovávejte jednoduše – administrátor, editor, fakturant, účetní, operátor výroby – tak, aby odpovídaly reálným rolím, na které je zákazník zvyklý.

2. **Definice cíle** – tato část je hlavní – zde je nutno popsat, co chcete se systémem vykonat za činnost, jedná se o samotnou specifikaci úkolu, tedy o specifikaci uživatelského příběhu. Snažíme se, aby popis byl výstižný, aby věcně a přímo popisoval specifikaci úkolů, ale aby se na druhou stranu nerozvíjel do epické šíře. To nechceme. Jednak proto, že takový popis velmi často obsahuje nejednoznačnosti, jednak proto, že je obtížnější na pochopení", a jednak proto, že potom musíme číst dlouhé texty, což nás zdržuje. Krátce a stručně – tohle motto bychom si měli vzít k srdci.
3. **Definice užitku** – poslední je definice užitku – čeho vlastně chceme dosáhnout. Ve skutečnosti se jedná o nepovinnou část uživatelského příběhu. Pouze tehdy, cítíte-li, že by bylo vhodné tuto informaci doplnit, můžete. Ale uživatelský scénář bude v pořádku i bez této informace.

A jak může vypadat reálný uživatelský příběh? Například takto:

Hledání uživatelů
Jako uživatel chci, abych mohl vyhledávat ostatní uživatele podle jména a příjmení.

Přidání uživatele
Jako administrátor chci, abych mohl přidat nového uživatele.

Výpis uživatelů
Jako administrátor chci, abych mohl vypsat na obrazovku seznam uživatelů tak, aby bylo možné najít duplicity.

Tip: K čemu nám vlastně slouží definice užitku. Je to dodatečná informace. Podívejte se na poslední uživatelský příběh. My z něj vidíme, proč vlastně chce administrátor výpis. Chce jej proto, aby mohl hledat duplicity. To znamená, že bude vhodné funkcionalitu naprogramovat tak, aby se k tomuto účelu co nejlépe používala. Tedy tohle je pravý cíl dodatečné informace – upřesnit, k čemu chceme funkcionalitu využívat, aby byla opravdu přizpůsobena. Zkuste si představit, že budete chtít auto. Ale určitě dodáte, k čemu jej budete používat. A tato informace bude sloužit k tomu, aby vám kdokoliv mohl rozumně poradit, jaké auto by pro vás bylo nejlepší.

Upozornění: Určitě je vhodné zmínit, že uživatelské příběhy nemusí mít přesně tuhle danou podobu. Existuje více šablon, jak tvořit uživatelské příběhy. Například:

Abych dosáhl <užitku> jako <role>, chci <cíl>

Abych mohl najít duplicity jako administrátor, chci vypsat na obrazovku seznam uživatelů.

nebo

Jako <kdo> <kdy> <kde>, chci <co>, protože <proč>

Jako administrator kdykoliv a kdekoliv chci vypsat seznam uživatelů na obrázku, protože potřebuji hledat duplicity.

Zcela úmyslně jsem, byť za cenu jisté šroubovanosti, přepsal jeden z výše uvedených uživatelských příběhů. Je vidět, že různé formulace jsou vzájemně ekvivalentní či alespoň velmi podobné. Je proto na vás, který konkrétní způsob zvolíte. Je dokonce možné kombinovat několik způsobů, nicméně měli byste vždy zajistit, aby existoval určitý pořádek.

Uživatelské příběhy přitom mohou vypadat jako poměrně nedostatečný popis požadavků. A je nutné uznat, že v tradičních metodikách by se skutečně o nedostatečný popis jednalo. Protože tam předpokládáme, že nejprve je vytvořena specifikace, která je schválena jako dokument, a poté se teprve jde programovat. Důvody, proč v takovém případě musí být požadavky specifikovány detailně, jsou v zásadě dva:

1. Specifikace je na úrovni kontraktu, tudíž musí být vytvořena tak, aby opravdu bylo možné kontrakt dodržet a vyhodnotit.
2. Specifikace je pro programátory jediným podkladem, oni se zákazníkem již nevyjednávají, proto je nutné, aby taková specifikace byla co nejpřesnější.

Tohle ve SCRUMu odpadá, protože

1. Specifikace není kontrakt, ve SCRUMu se snažíme vyhnout tvrdým smlouvám všude, kde to jen jde. Cílem je nikoliv otrocké dodržení nějaké specifikace, ale vytvoření funkčního kvalitního softwaru. Stejně se počítá s mnoha změnami, úpravami a dodatečnými požadavky.
2. Komunikace se zákazníkem nikdy nekončí, navíc se počítá se zákazníkem na pracovišti co možná nejčastěji, takže je možné se kdykoliv doptat na potřebné další informace, je možné zkoušet různé varianty, které zákazník okamžitě může testovat.

Upozornění: Uživatelské příběhy jsou jednoduché, ale samy o sobě postrádají jednu důležitou věc. Neříkají, jak se má jednou, až bude příběh implementován, testovat. To je důležité, protože jen tak můžeme zajistit, že tvoříme to, co skutečně máme tvořit. Proto se každý uživatelský příběh spojuje s tzv. akceptačním kritériem. Tento cizí termín ve skutečnosti jen říká – až bude jednou scénář implementován, provedeme tu a tu operaci. A pokud je vše v pořádku, systém bude mít takový a takový výstup. Například chceme zmíněný výpis. Akceptačním kritériem může být skutečnost, že zkusíme výpis provést nad známou množinou dat a zkontrolujeme, zda ve výpisu skutečně jsou všechny záznamy.

Uživatelské příběhy mají přitom několik velmi důležitých pozitiv, která mohou sloužit jako shrnutí této kapitolky:

- Extrémní stručnost – přitom reprezentují malé části obchodní hodnoty, které programátor může realizovat v období několika dnů až týdnů. Každá uživatelský scénář je jeden úkol a každý úkol je část pracovního zatížení zákazníka. Splněním příběhu umožníme zákazníkovi, aby provedl jeden z úkonů, které jsou nutné pro jeho práci.
- Uživatelské příběhy umožňují vývojářům i zákazníkům snadno řešit požadavky po celou dobu životnosti projektu – pokud přijde nový úkol, který je třeba do systému doprogramovat jako novou funkcionalitu, pak prostě definujeme další uživatelský příběh a tento následně implementujeme.
- Potřebují velmi malou údržbu – příběhy nevyžadují složitou strukturu a složitou správu tak, jak to známe z jiných metodik vývoje softwaru, kde správa požadavků je samostatným vědním oborem. Každý takový příběh přitom využíváme pouze v okamžiku, kdy na něm pracujeme – to snižuje provázanost a tedy riziko chyb.
- Umožňují rozdělení projektů do malých částí – každá část představuje implementace funkcionalit, které pokrývají určitou sadu uživatelských příběhů.
- Uživatelské příběhy jsou vhodné pro projekty, které mají proměnné nebo špatně pochopitelné požadavky: iterace umožňují celou situaci řídit a postupně upřesňovat požadavky až do okamžiku, kdy zákazník považuje softwarový produkt, který je vyvíjen, za uspokojivý.