AGILNÍ METODIKY VÝVOJE SOFTWARE

Postupy předchozích metodik, založené na důsledné analýze a propracovaném návrhu jsou obecně nejlepší.

Ale…

„Děláte web půl roku? Konkurence mezitím spustila dva…“

Zdánlivě to může vypadat tak, že neexistence zákazníkovy představy, co vlastně chce je výhodou – „něco“ mu dodáme a zákazník bude spokojen.

Zákazník sdělí na konci projektu, že výsledek není to, co chtěl.

Chce projekt dodělat/předělat – za původně dohodnutou cenu.

Svět se mění – zákazník očekává kvalitu, ale není ochoten na ni dlouho čekat.

Tento rozpor se snaží řešit agilní metody snahou o **užší sepětí zákazníka s vývojovým týmem**.

2004 „MANIFEST AGILNÍHO VÝVOJE SOFTWARU“

Jedinou cestou, jak prověřit správnost navrženého systému, je co nejrychleji jej vyvinout, předložit zákazníkovi a na základě zpětné vazby upravovat.

Tradiční přístup - požadavky jsou stanoveny na začátku vývojového procesu a jsou v průběhu vývoje neměnné. Proměnné jsou zdroje a čas. Kontakt se zákazníkem je pouze ve fázi analýz a pak až následuje předvedení hotového produktu nebo prototypu. V průběhu řešení není kontakt s uživatelem.

Agilní přístup považuje za neměnné zdroje a čas, předmětem změn je funkcionalita. Na počátku projektu se stanoví nejdelší možný čas a náklady. Tým v průběhu řešení komunikuje se zákazníkem a průběžně přehodnocuje priority.

TEZE AGILNÍHO MANIFESTU

- Přijmout a umožnit změnu je efektivnější než se změně bránit
- Je třeba být připraven na nepředvídané události – „jedinou jistotou na projektu je změna“.

PRINCIP AGILNÍCH METOD

- Užitná hodnota pro zákazníka
- Změny jsou výhodou (pro zákazníka může být konkurenční výhodou, agilní metodiky neřeší nic, co není momentálně potřebné, protože v budoucnu se to může změnit)
- Časté dodávky (velmi krátké iterace)
- Klíčová je motivace
- Zákazníci spolupracují s týmem
- Úspěch posuzujeme podle fungování a ne podle splnění specifikace
- Zásadní je jednoduchost
- Důvěra a komunikace vedou ke kreativitě
- Jak zvýšíme efektivitu?
- Perfektní návrh a řešení (změna není považována za důkaz jeho nekvality)
- Udržitelný vývoj (přesčasy a práce v noci řeší krátkodobě problémy, ale dlouhodobě snižují produktivitu práce)

TÝM AGILNÍHO VÝVOJE

- Do 10 členů:
  - Kouč
  - Programátoři
  - Časoměřič
  - Stále přítomný pracovník uživatele (asi nejde vždy dodržet, mohou být třeba různí pracovníci, daný pracovník je nepostradatelný)
  - Programátoři pracují ve dvojicích, které se mění
    - První programátor – vymýšlí a píše
    - Druhý programátor – oponuje, kontroluje, spoluvymýšlí

- Místnost pro odpočinek a jednání
- Důraz na využití kreativity
- Dokumentace – minimalizovat (nikdo nečte), jen přehledný zdrojový kód
- Přesčasy dlouhodobě nezvyšují produktivitu práce

Agilní vývoje omezuje

- rizika spojená s nepřesným zadáním nebo se složitostí budovaného systému
- rizika spojená s fluktuací členů týmu,
- rizika spojená s tím, že neexistuje dokumentace v obvyklém rozsahu,
- rizika spojená s nedodržováním termínů a překračováním rozpočtů.

Kdy není vhodné používat agilní metodiky:

- Kritické systémy, kde je nutné přesně dodržovat dohodnuté (technologie)
- Rozsáhlé systémy, které se nedají dobře dekomponovat
- Nejsou k dispozici kvalitní řešitelé
- Není ochota se domlouvat o cíli za pochodu (jak uzavřít smlouvu, jak sankce za neplnění)

PŘEHLED AGILNÍCH METODIK

- Adaptivní vývoj softwaru
- Extrémní programování
- Lean Development
- SCRUM
- Crystal metodiky
- Test-Driven Development
- Feature Driven Development

SCRUM

- Název pochází z rugby – mlýn hráčů za účelem společného dotažení míče na pozici
- 1995 Schwaber, Beedle
- Vychází z objektově orientovaného přístupu
- Vývoj v posloupnostech, intervalech, které se nazývají „**sprinty**“ (max. měsíc)
- Krátké **denní Scrum Meetings** -> konkrétní určení činností; které položky z minulého meetingu byly dokončeny a jaké nové úkoly vznikly; předvedení výsledků zákazníkovi
- Tyto schůzky jsou zásadní, určují:
    - Shrnutí dosavadního pokroku
    - Předvedení mezivýsledků
    - Identifikace nových úkolů
    - Zvyšování soudržnosti týmu
    - Odhalování problémů v mezilidských vztazích
- SCRUM počítá s tím, že nelze naplánovat přesný průběh vývoje a proto se o to nepokouší, přesné plánování suplují denní úkoly

Charakteristika projektů podle SCRUM:

a) Flexibilní předměty dodání (obsah dodávky diktován prostředím. Např. co má být výsledkem analýzy? Někdy je lepší specifikace podle norem, někdy je lepší objektový model, jindy dodání prototypu…)
b) Flexibilní harmonogram (flexible schedule) – dodání může proběhnout později, než se očekávalo, ale zákazník s tím musí být ihned srozuměn (toto je obtížné, záleží na úrovni vztahů se zákazníkem, nebezpečí, že to bude vnímat jako problém nebo projev neprofesionality)
c) Malé týmy – ideální 3 – 6 lidí, na jednom projektu může pracovat více týmů.
d) Časté revize
e) Spolupráce – intenzivní komunikace členů týmu, týmu se zákazníkem i zadavatelem

Backlog – informace o vlastnostech, funkcích a činnostech, které je třeba implementovat (může být forma tzv. „User Stories“). Seznam úkolů v Backlogu může modifikovat pouze manažer projektu (Scrum Master), ostatní pouze čtou. Úkoly jsou na začátku ve spolupráci se zákazníkem seřazeny podle priority.

Riziko – silný důraz na analýzu rizik. Revize rizik na konci každé interakce, ale i v průběhu každodenních schůzek

Sprint – základní vývojová entita (iterace) – fáze vývoj (Develop), zabalení (Wrap), revize (Review), přizpůsobení (Adjust); obvykle 30 dní

LEAN DEVELOPMENT

Původně Toyota - pointou je **odstranění všeho zbytečného a minimalizace zásob** (nic nesmí ležet na skladě); později transformováno do oblasti SW

Pravidla Lean Development:

1. odstranit vše, co je zbytečné
2. minimalizovat zásoby (minimalizovat meziprodukty)
3. maximalizovat tok (=zkrátit čas potřebný na vývoj)
4. vývoj je tažen poptávkou (rozhodnutí dělat čím jak nejpozději je to možné)
5. pracovníci mají pravomoci rozhodovat
6. hlavním cílem je uspokojovat požadavky zákazníků (teď i v budoucnu)
7. zpětná vazba (nebát se změn v učiněných rozhodnutích)
8. odstranit lokální optimalizaci (neustálé optimalizace stávajícího řešení nemají smysl)
9. vybudovat partnerství s dodavateli
10. vybudovat prostředí pro možnost neustálého zlepšování

Pokud věc nepřidává novou hodnotu, je to zbytečnost.

LD stanovuje **šest druhů plýtvání**:

1. nadvýroba (u sw nadbytečné požadavky, které pak nejsou používány)
2. čas tracený čekáním (tester čeká na dokončení funkcí – mezitím může dělat něco jiného)
3. plýtvání související s přepravou (u sw spočívá řešení v automatizaci některých procesů)
4. plýtvání související se zpracováním (vedení týmu by mělo mít přehled kdo co dělá a v jaké fázi se vývoj nachází)
5. nefektivní práce (u sw využití existujících sw nástrojů a řešení, vývojáři o nich často ani neví)
6. defekty ve výrobcích (program bez chyb téměř neexistuje, ale je třeba přijmout opatření k minimalizaci chyb)

Vývojáři musí v každém případě chápat, jak vlastně přispívají ke konečnému cíli a k čemu jejich práce vede!

Uspokojení potřeb uživatele – ten často neví co chce, požadavky se průběžně mění a občas jsou zdánlivě nesmyslné – je lépe pracovat na partnerství a užším vztahu se zákazníkem než ho na začátku nutit podepsat stostránkovou funkční specifikaci.

FEATURE DRIVEN DEVELOPMENT

Hlavní roli hrají vlastnosti výsledného produktu, **vlastnosti produktu řídí vývoj**

90. léta 20. století

Založeno na iterativním vývoji, krátké iterace;

Modelování – vývoj začíná vytvořením **globálního modelu** systému – z něho by mělo být patrné celkové směrování vývoje – cíl není přinést kompletní přehled funkcí, ale naznačit, z čeho se bude systém skládat a jak bude komunikovat s okolím.

Předpokládá se, že zákazníkovi jsou neustále dodávány beta verze (minimálně každých 1-3 týdne). Zákazník vidí, že vývoj postupuje vpřed (psychologický efekt) a taky má možnost do vývoje zasahovat.

Vlastnost (feature) – malý výsledek (funkčnost) užitečná z pohledu zákazníka

Vlastnost je charakterizována:

- Měřitelnosti (Je implementovaná funkce totožná s funkcí požadovanou zákazníkem?)
- Srozumitelností (Musíme být schopni vlastnost popsat)
- Realizovatelností (Lze vlastnost dodat? Nebude její vývoj trvat příliš dlouho?)

Postup:

1. Vytvoří se seznam vlastností.
2. Seřadí se podle priority.
3. Nad seznamem probíhá vývoj – průběžně vznikají a zanikají malé týmy mající na starost implementaci konkrétní vlastnosti.
4. Po implementaci nastupuje fáze testování a integrace.

Metodika je vhodná pro menší projekty.

TEST DRIVEN DEVELOPMENT

Základní myšlenka: testovací kód musí být připraven a dokončen **před** začátkem psaní kódu

1. napíšeme nový test požadované funkce tak, aby selhal
2. začleníme do projektu (do kompletní testovací sady) a ověříme, že selže
3. implementujeme požadovanou funkci
4. znovu spouštíme testy, jestliže není test úspěšný, musíme kód opravit
5. pokud test projde, zařadíme ho do testovací sady (knihovny)

Testování -> implementace -> návrh

Chyby jsou odchyceny při průchodu testovacím případem.

Tato metodika je méně procesně orientovaná, nezabývá se tvorbou specifikací.

Výhoda

kvalitní software s předvídatelným a dobře prozkoumaným chováním

Nevýhoda

nutnost pevné ruky ze strany řízení projektu (pro programátory nepohodlné psát testy na počátku)