INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

VYSOKÁ ŠKOLA BÁŇSKÁ – TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA STROJNÍ

INFORMAČNÍ SYSTÉMY

Vývoj IS – Softwarové inženýrství

Ing. Roman Danel, Ph.D.

Ostrava 2013

ISBN 978-80-248-3051-3

Tento studijní materiál vznikl za finanční podpory Evropského sociálního fondu (ESF) a rozpočtu České republiky v rámci řešení projektu: CZ.1.07/2.2.00/15.0463, MODERNIZACE VÝUKOVÝCH MATERIÁLŮ A DIDAKTICKÝCH METOD

OBSAH

1 VÝVOJ IS – SOFTWAROVÉ INŽENÝRSTVÍ

OBSAH KAPITOLY:

Problémy a specifika vývoje software

Odlišnost vývoje SW oproti ostatním oborům

Přístup k vývoji IS

Příklad agilní metodiky vývoje - SCRUM

MOTIVACE:

V této kapitole se budeme věnovat problémům spojeným s vývojem informačních systémů. Seznámíte se s disciplínou nazývanou softwarové inženýrství, jejímž předmětem jsou metodiky pro tvorbu software. Vysvětlíme si rozdíl mezi úkolocentrickým a hodnotocentrickým přístupem k tvorbě software a seznámíme se s nejčastěji používanými metodikami vývoje software. Vysvětlíme si rozdíl mezi klasickým a agilním přístupem k řízení projektu vývoje.

CÍL:

Po prostudování kapitoly budete znát základní nejčastěji používané metodiky vývoje software; a to skupinu metodik klasických jako je například vodopád, spirála či RUP a skupinu metodik agilních, jako např. SCRUM nebo test-driven development.

1.1 PROBLÉMY A SPECIFIKA VÝVOJE SOFTWARE

Vývoj prvních programů byl prováděn nadšenci, programy byly šité na míru. Žádná metodika vývoje SW v té době neexistuje.

Vývoj SW byl vnímán jako výzkum. Cíl, co bude software dělat, si určoval programátor sám na základě velmi vágního zadání (ostatní často neměli představu, co by od počítačů a software mohli chtít). Vývoj většinou nebyl omezen žádnými termíny (až byl software hotov, došlo k jeho nasazení).

V důsledku stále většího rozmachu informatiky a počtu projektů, které neskončily úspěšně (ve smyslu – nebyly dokončeny včas nebo byly příliš nákladné nebo nebyly dokončeny vůbec) se na přelomu 60. a 70. let 20. století se začalo mluvit o tzv. „softwarové krizi“.

1.2 SOFTWAROVÁ KRIZE

- Neúnosné prodražování projektů
- Neúnosné prodlužování projektů
- Nízká kvalita programů
- Nízká produktivita programátorů
- Neefektivita vývoje
- Nejistota výsledku

„Softwarová krize“ vyvolala potřebu používání metodik pro řízení vývoje software (IS).

Jak vypadají problémy softwarové krize z dnešního pohledu?

1.2.1 Problém – špatná komunikace

Jedním z hlavních problémů je špatná komunikace. Dříve často zákazník jednal přímo s programátorem.

Dnes: tendence k oddělení funkce analytika od vývojáře (kodéra).

1.2.2 Problém – nesprávný přístup

Problém přístupu lidí k vývoji. Programátoři občas sklouzávají k tendenci předvést se, seberealizovat, „vyřádit se“.

Dnes: zaměření na týmovou spolupráci, důležitá je spokojenost zákazníka.

1.2.3 Problém – špatné plánování

Je obtížné vypracovat plán vývoje, který je přijatelný pro zákazníka a realizovatelný pro vývojáře.

Představa typu „ nějak se to stihne“.

Po zadání projektu někdy následovalo bezprostřední „bušení do klávesnice“.

Dnes: metodiky vývoje software

1.2.4 Problém – nízká produktivita

Programátoři se zabývají vším možným jen ne tím, co bylo potřeba. Tendence psát kód okamžitě, vychrlit čím jak nejvíce řádků kódu.

Dnes: tým s přidělenými rolemi. Důraz na koordinaci vývoje. Vývoj podle předem stanovených specifikací.

1.2.5 Problém: podcenění hrozeb a rizik

Problémy, které by mohly být vyřešeny hned na začátku, byly přehlíženy. „To vyřešíme nakonec“, „to nebude problém“, „to se nepozná“.

Dnes: snaha odhalit chyby na začátku. Metodiky na provádění analýzy rizik (rozbor potenciálních hrozeb). Použití metodik, které berou riziko jako jednu ze základních jednotek.

1.3 ODLIŠNOST VÝVOJE SW OPROTI OSTATNÍM OBORŮM

Čím je vývoj softwaru specifický oproti ostatním oborům?

1.3.1 Kvalita odborných pracovníků je pro úspěch rozhodující

V klasických oborech je strůjcem úspěchu management, pracovníci na nižší úrovni jsou považováni za snadno zastupitelné a nahraditelné. V oblasti vývoje software to tak úplně neplatí a odchod klíčového programátora z týmu může způsobit vážné komplikace (nebo dokonce krach projektu).

Velký rozdíl je v produktivitě. Programování **není jen o zvládnutí programovacího jazyka, ale i o zkušenostech**. Mezi programátory jsou zásadní rozdíly dané vzděláním, zkušenostmi a talentem.

1.3.2 Odborní pracovníci se musí neustále zdokonalovat a vzdělávat

Nové technologie přicházejí v IT **cca co tři roky**.

Neustálé školení a vzdělávání je nutnost. Programátorům hrozí technologické zaostání a na druhé straně, díky rychlému tempu, také vyhoření.

1.3.3 Výhodný je jiný způsob práce

Programování není o osmihodinové pracovní době.

Programátory často práce baví a je výhodné pracovní dobu pružně měnit. V týmu prodchnutém nadšením lze dosáhnout značné produktivity, nesmí ale jít o práci ve stresu a období zvýšeného úsilí musí být časově omezeno.

1.3.4 Nejproduktivnější jsou malé, pozitivně motivované týmy

Striktně hierarchický systém, s prací pod tlakem, že nebudou prémie, v softwarovém vývoji nefunguje.

Vývoj software je tvůrčí práce. Manažer vývoje by neměl vývoj řídit svými příkazy, jestliže tým úspěšně funguje. Při vzniku problémů by naopak měl zasáhnout čím jak nejdříve.

1.3.5 Chyba je zákonitou součástí vývojového procesu

Vývoj SW je tvůrčí a kreativní činnost, nelze ji naplánovat jako stavbu budovy.

I v nejlépe řízených projektech se vyskytnou chyby. Je ale třeba stanovit takové postupy, aby se chyby maximálně eliminovaly.

1.3.6 Prudký technologický rozvoj přináší nové příležitosti

Rutinéři na vedoucích místech je často neumí rozpoznat. Iniciativa může přicházet zdola. Úspěchy v minulosti nic neznamenají.

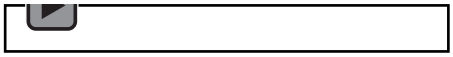
Říká vám něco jméno Digital Equipment Corporation (DEC)?

(V 20. století spolu s IBM největší IT firma na světě, přes 50 tisíc zaměstnanců, jeden z leaderů na poli mainframe; firma, která přinesla úspěšné řady mainframe počítačů, jako byly PDP, VAX, ALPHA a operačními systémy RSX a VMS. V roce 1998 byla firma spojena s firmou Compaq a dnes je to jen divize v koncernu Hewlett Packard.)

1.4 NEJČASTĚJŠÍ PROBLÉMY VÝVOJE SW

- Zpoždění
- Vysoká chybovost
- Neplnění požadované funkčnosti
- Nedostatečná výkonnost
- Složité uživatelské rozhraní
- Obtížná udržovatelnost programu

Audio 3.1 Nejčastější problémy vývoje IS



1.5 ZÁKLADNÍ PŘÍČINY PROBLÉMŮ

- Podcenění projektu a špatný odhad (čas, náklady)
- Špatné zadání
- Nedostatečná analýza
- Přílišná složitost projektu
- Přehnaný důraz na technologii (použití novinek bez zkušeností)
- Špatná kvalita programového kódu (chybový, nesrozumitelný, pomalý, nedostatečně komentovaný)
- Nevhodné metodiky, postupy, technologie
- Nedostatečné testování
- Špatné projektové řízení

1.6 ÚVOD DO SOFTWAROVÉHO INŽENÝRSTVÍ

Předmětem softwarového inženýrství jsou metodiky pro řízení vývoje softwaru.

- Proč potřebujeme tyto metodiky?
- Čím je vývoje softwaru specifický oproti jiným odvětvím?

1.6.1 Softwarové inženýrství

Softwarové inženýrství je **zavedení a používání inženýrských principů tak, abychom dosáhli ekonomické tvorby softwaru**.

Takto vytvořený software je spolehlivý a pracuje na dostupných výpočetních prostředcích.

1.7 PŘÍSTUP K VÝVOJI IS

Z pohledu tvůrců informačních systémů můžeme přístup k vývoji IS rozdělit do dvou oblastí:

a) Úkolocentrický
b) Hodnotocentrický

1.7.1 Úkolocentrický přístup k vývoji IS

Úkolocentrický (neboli work-down) přístup k vývoji IS je založen na základech **projektového** řízení. Vývoj a implementace IS postupuje podle daného plánu (harmonogramu) stanoveného na začátku řešení projektu, který se rozpadá do řady plnění dílčích konkrétních úkolů (termínů). Úkoly jsou postupně realizovány, plnění harmonogramu je průběžně kontrolováno.

Úkolocentrický přístup je vhodný u projektů s dobře známým návrhem a nízkými riziky -cíl je jasně definován, řešení má stanovenou koncepci.

Hodnota, kterou IS zákazníkovi přinese, je jednoznačně stanovena na začátku projektu.

Velkou výhodou tohoto přístupu je kontrola a přehled nad postupem prací, projekt je řiditelný podle jasně stanovených kritérií.

Nevýhody

- Změny vzniklé v průběhu řešení projektu se obtížně integrují
- Nápady a postupy, které by pro zákazníka mohly znamenat přínos, které však nebyly v původním návrhu, nemohou být uplatněny (cílem je dodržet harmonogram a rozpočet)
- Nedostatečná analýza a s tím vzniklá neurčitost řešení může způsobit vážné komplikace
- Zákazník má v průběhu projektu minimální vliv na řešení projektu
- Kvalita je definována splněním specifikace (dodržením harmonogramu)

1.7.2 Hodnotocentrický přístup k vývoji IS

U hodnotocentrického (value-up) přístupu vývoj probíhá v iteracích se zákazníkem. Je kladen důraz na hodnotu IS pro zákazníka, která se může průběžně zvyšovat. I při hodnotocentrickém přístupu je vytvářen plán, ale postup podle harmonogramu není hlavním cílem; objeví-li se během vývoje skutečnosti a postupy, které vedou k tomu, že zákazník dostane více, má tato inovace přednost před dodržením harmonogramu za každou cenu. Části SW, které se v průběhu řešení ukážou jako nepotřebné nebo zastaralé, mohou být z projektu zrušeny. Snahou je dodat zákazníkovi čím jak nejdříve fungující části tak, aby zákazník mohl řešení ověřit a připomínkovat a jeho podněty mohly být do vývoje zapracovány.

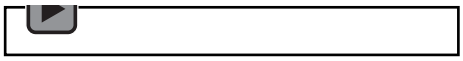
Hodnotocentrický přístup má:

- Zaměření na průběžné vytváření hodnoty
- Zapojení zákazníka do řešení projektu
- Očekáváme nejistotu a jsme na ni připraveni
- Zdrojem hodnoty jsou jednotlivci – uznáním zvyšujeme kreativitu
- Výkon povzbuzujeme skupinovou zodpovědností za výsledku a efektivitu týmu

- Kvalita je definována přínosem pro zákazníka (termín může být posunut, pokud to zákazníkovi něco přinese)
- Změny a odchylky jsou brány jako přirozená součást řešení
- Lze využít kreativity řešitelů

Samozřejmě i tento přístup má své nevýhody – obtížnější řiditelnost projektu, včetně vyhodnocení postupu prací, nebo hrozba, že projekt nebude uspokojivě ukončen apod. Tento postup je také méně vhodný u velkých investičních akcí, kde investor a uživatel systému jsou různé subjekty.

Audio 3.2



1.8 DOPORUČENÍ K VÝVOJI IS

Jedním z důležitých kritérií úspěchu IS jsou:

- snadnost ovládání
- rychlost uvedení na trh

Je výhodnější napsat aplikaci, která bude rozšiřitelná v budoucnu než monolitický systém, který bude obsahovat všechny možné funkce hned na začátku (včetně funkcí, které nikdy nikdo nepoužije).

Aplikace psaná na míru konkrétním požadavkům, postrádající jakékoli rysy obecnosti, může být v budoucnu obtížně modifikovatelná.

Tendence psát příliš obecné systémy často vede ke dvěma výsledkům – nesrozumitelné a neprůhledné architektuře nebo k systémům, které umožňují díky množství nastavitelných parametrů vyvolat chování, jenž ani tvůrce systémů není schopen dokumentovat.

Je tedy třeba najít rozumný kompromis mezi obecností a konkrétními požadavky.

Přístup k vývoji je odlišný, pokud vytváříme jednorázovou aplikaci, která bude fungovat po omezenou dobu, nebo se snažíme o vývoj „krabicového software“, který bude fungovat v mnoha firmách. V druhém případě určitě stojí za to navrhnout celou koncepci systému tak, aby byl maximálně obecný a parametrizovatelný. V případě jednorázového určení může být snaha po přílišné obecnosti ztráta času a prostředků.

1.9 CO JE TO METODIKA?

Metodika vývoje SW zahrnuje všechny etapy řešení, měla by odpovídat na otázky.

Proč? Kdo? Kdy? Co?

Metodika je tedy souhrn postupů vedoucích k dodání funkčního software.

1.10 KLASIFIKACE METODIK

Významné tradiční metodiky návrhu a vývoje IS:

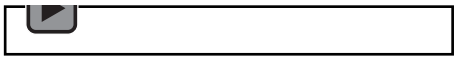
- Model „napiš – oprav“ (Build and Fix)
- Striktní posloupnost fází (Stagewise)

- Vodopádový model (Waterfall)
- Spirálový model
- Další metodiky: RUP, UP, USDP, …

Některé agilní metodiky vývoje IS:

- Extrémní programování
- Crystal
- SCRUM
- Aspect Oriented Programming
- Test Driven Development
- Lean Development

Audio 3.3 Klasifikace metodik



1.11 TRADIČNÍ METODIKY

1.11.1 Model „Napiš – oprav“ (Build and Fix)

Implementace -> Dodání -> Opravy chyb

Tato metodika je nejstarší známou metodikou z oblasti vývoje SW, zde je uvedena jen pro úplnost a s výjimkou malých jednorázových projektů je její použití nevhodné.

1.11.2 Stagewise model

Tento model vývoje software byl definován v roce 1957.

Založen na striktní posloupnosti fází:

- Definice problému
- Analýza
- Specifikace požadavků
- Návrh
- Architektura
- Implementace (+testování)
- Provoz

Metodika Stagewise je založena na následujících skutečnostech:

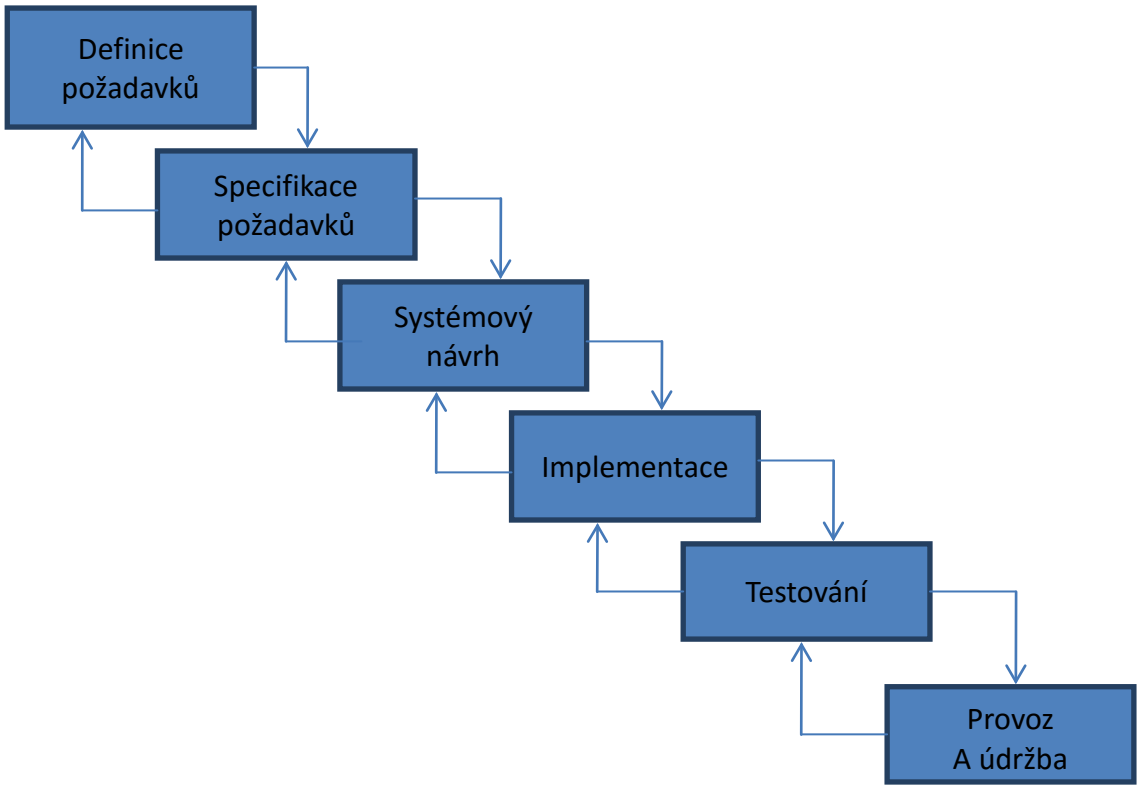
Absence zpětné vazby, neprovádí se revize žádné fáze, nerevidují se požadavky ani se nehledají rizika.

1.11.3 Model vodopád (Waterstage)

- 1970 Winston Royce
- Každá etapa má stanovený přesný cíl a dokumenty, které musí v jejím průběhu vzniknout
- Na konci každé etapy dochází k jejímu vyhodnocení a případně přepracování nebo opravení
- Možnost vrátit se zpět do předchozí etapy
- Pokračuje se teprve tehdy, je-li etapa zcela dokončena a schválena (pak již návrat není možný)

Výhody:

- Jednoduchý
- Ideální pro řízení
- Vnáší disciplínu do vývoje



Nevýhody:

- Dodávka formou „velkého třesku“
- Určitá nepružnost
- V době mezi analýzou a nasazením se mohou změnit požadavky

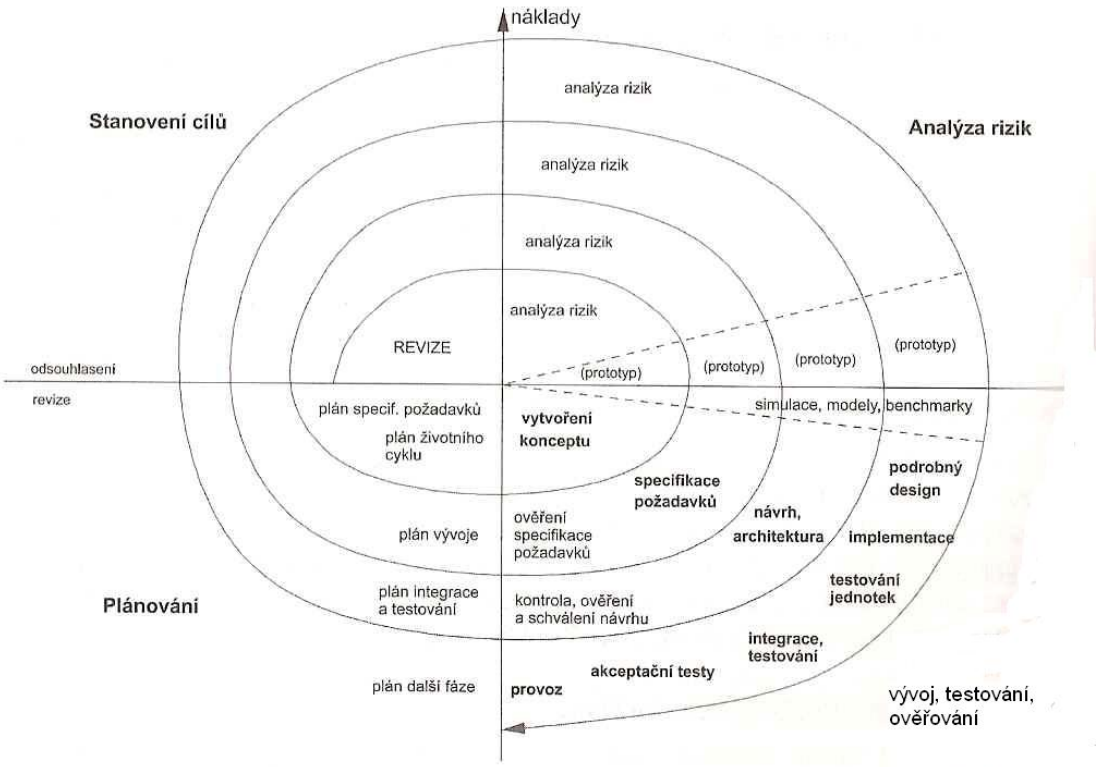
„Začnu-li padat, nezastavím se dříve, než se rozbiji o kámen zvaný předvedení.“

1.11.4 Spirálový model

- 1985, Barry Boehm
- Zavádí iterativní přístup a opakovanou (důslednou) analýzu rizik

Rizika – situace nebo události, které mohou způsobit nesplnění cílů projektu.

Vychází se z předpokladu, že na začátku je obtížné nebo až nemožné přesně specifikovat všechny funkce.



Obr. 3.1 Model „spirála“ [Zdroj: http://ari.wikidot.com/zivotni-cyklus-is]

Ve spirálovém modelu se tedy stanoví jen obecný rámec.

Vyvine se část aplikace -> předvedení a konzultace se zákazníkem -> pokračuje se dalším krokem.

Vývoj probíhá ve spirále, v postupných iteracích. Na rozdíl od dále zmíněných agilních metodik ale zákazník vidí pouze jednotlivé výsledky a nepodílí se průběžně na vývoji SW.

Výhody:

- Vytváří prostředí pro vývoj znovupoužitelných komponent
- Je komplexní a vhodný i pro složité projekty (díky důrazu na plánování)
- Včasné vyloučení nevhodných řešení

Nevýhody:

- Celková komplikovanost
- Software není uvolněn před dokončením posledního cyklu
- Změna požadavků je možná pouze po dokončení cyklu
- Pro nové druhy aplikací (např. internetové) je nepružný

Je vhodnější pro rozsáhlé projekty!

1.11.5 RUP – Rational Unified Process

- Vývoj probíhá v iteracích; dělí se na 4 fáze: zahájení, projektování, realizace, předání (zákazníkovi nebo do další fáze vývoje)

- Robustní, propracovaná metodika, vhodná pro větší projekty a rozsáhlejší týmy
- Komerční produkt (dříve firma Rational, dnes IBM)
- Je založen na využití UML (grafický modelovací jazyk používaný pro návrh systémů, vychází z objektového přístupu, pomocí sady diagramů modeluje statický i dynamický pohled na systém)

Klíčové principy:

- Iterativní vývoj softwaru (vychází ze spirálového modelu, průběžná detekce rizik)
- Správa a řízení požadavků (požadavky se v čase mění)
- Použití komponentové architektury
- Vizuální modelování softwaru (za účelem porozumění systému; využití UML)
- Průběžné zajišťování a ověřování kvality (po předání je nalezení problému dražší)
- Řízení změn (počítáme s tím, že změny nastanou, neřízení změn vede k chaosu)

Výhody RUP:

- Obecnost a mohutnost
- Iterativní přístup – včasné odhalení rizik
- Snazší správa změn
- Provázanost s notací UML, dokumentace
- Výrobce průběžně pracuje na zlepšování metodiky
- Existence doplňkových nástrojů

Nevýhody RUP:

- Komerční, tj. placený produkt
- Rozsáhlost RUP může být na škodu u malých týmů – tým stráví spoustu času implementací metodiky
- Její použití vyžaduje hluboké studium, týká se i projektových manažerů

Klasické metodiky vývoje SW jsou tedy založeny na **striktní definici postupů**.

1.11.6 Unified process

Obdobou RUP je metodika UP (Unified Process), jedná se o její nekomerční verzi.

Unified Process je řízena požadavky a riziky. Je iterativní a inkrementální. Každá iterace zahrnuje všechny fáze běžného vývoje (plánování, analýza, konstrukce, integrace, testování).

Jedna iterace zahrnuje:

1. Zahájení – stanovení proveditelnosti, vytvoření business případu, zachycení požadavků, identifikace rizik
2. Rozpracování – spustitelná verze architektury, zpřesnění rizik a požadavků, definice kvality, plán konstrukce a prostředků
3. Konstrukce – doladění požadavků, implementace
4. Zavedení – opravy chyb, příprava pracoviště k nasazení, přizpůsobení SW a pracoviště, manuály, konzultace

1.12 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**.

1.12.1 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.

Hlavní myšlenkou agilního přístupu je tedy vtáhnout zákazníka do procesu vývoje.

Tradiční přístup - požadavky jsou stanoveny na začátku vývojového procesu a jsou neměnné. Proměnné jsou zdroje a čas.

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

1.12.2 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“.

1.12.3 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)

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

1.12.4.1 Agilní vývoj 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

1.12.4.2 Kdy není vhodné používat agilní metodiky:

- Kritické systémy, kde je nutné přesně dodržovat dohodnuté podmínky a postupy
- 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)

1.13 PŘEHLED AGILNÍCH METODIK

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

1.13.1 SCRUM

- Název pochází z ragby – mlýn hráčů za účelem společného dotažení míče na pozici
- 1995 Schwaber, Beedle
- Vývoj v posloupnostech, krátkých etapách, které se nazývají „**sprinty**“ (max. měsíc)
- Pro každou etapu je stanoven seznam úkolů neboli **backlog**
- Úkoly jsou seřazeny podle priority, určené ve spolupráci se zákazníkem
- 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“). Modifikovat může pouze manažer projektu, ostatní pouze čtou.

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.

1.13.2 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 (neustálá 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.

1.13.3 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. Podle seznamu 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.

1.13.4 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).

2 KONTROLNÍ OTÁZKY

1. Co je to softwarová krize a jaké jsou příčiny neúspěchu softwarových projektů?
2. Jaké jsou specifika vývoje SW oproti jiným odvětvím?
3. Jaký je rozdíl mezi úkolocentrickým a hodnotocentrickým přístupem k vývoji SW?
4. Na čem jsou založeny klasické metodiky vývoje SW?
5. Jaké klasické metodiky vývoje SW znáte?
6. Čím se liší agilní metodiky vývoje od klasických?
7. Popište základní principy agilní metodiky SCRUM
8. Jaké jsou výhody a nevýhody klasických a agilních metodik?

3 POUŽITÁ LITERATURA

[1] Guckenheimer, S. – Perez, J.: Efektivní softwarové projekty. Zoner Press, Brno 2007. ISBN: 978-80-86815-62-6.

[2] Paleta, P.: Co programátory ve škole neučí aneb softwarové inženýrství v reálné praxi. Computer Press, Brno 2003. ISBN: 80-251-0073-1.

[3] Kadlec, V.: Agilní programování – metodiky efektivního vývoje softwaru. ComputerPress, Brno 2004. ISBN: 80-251-0342-0.

[4] Agilní manifest, cz verze. [online] [cit 2013-05-19]. Dostupné na www http://agilemanifesto.org/iso/cs/.