Softwarový proces

- množina aktivit a (mezi)výsledků, jejichž výsledkem je softwarový produkt
- dle takto stanoveného předpisu se následně definují projekty (tzv. instanciace procesu), které jsou vztažené k jednotlivým zakázkám

Existují různé sw procesy, protože

- vytvářený sw je různorodý – pro různé typy systémů vhodné rozdílné procesy
- lidé mají různé znalosti, schopnosti a preference

Avšak ve všech typech sw procesů můžeme rozeznat různým způsobem organizované 4 základní aktivity:

- Specifikace sw – určení požadované funkčnosti a definice omezení
- Vývoj sw – tvorba sw splňujícího specifikaci
- Validace sw – ověření, že sw dělá, co potřebuje zákazník
- Evoluce sw – přizpůsobení sw měnícím se požadavkům zákazníka

Úroveň definice a hodnocení softwarového procesu je hodnocena dle stupnice SEI (Software Engineering Institute) a vyjadřuje vyspělost organizace z daného hlediska.

Tento model vyspělosti a schopností dodavatele se nazývá CMMI (Capability Maturity Model Integration, http://www.sei.cmu.edu/cmmi/), dříve CCM (Capability Maturity Model)

Modely sw procesu (-> modely životního cyklu sw projektu, metodiky):

- zobecněný popis procesu nebo části procesu, resp. popis procesu či jeho části z určitého pohledu

- tzn. existence řady modelů, žádný však dodnes nebyl přijat (a snad ani nebude) jako referenční (základem a „prarodičem“ všech vodopádový model)

- vhodně zvolený model může významně zvýšit výkonnost softwarového vývoje

- v jednodušších případech (neexistuje ucelený systém) nemluvíme o modelu, ale o pravidlech (často pokrývají jenom např. zadání projektu a jeho nasazení, vývojáři dodržují všeobecně uznávaná pravidla, tzv. best practices, mohou z nich vybírat)

- nemůžeme libovolně vybírat, co z modelu použijeme (můžeme samozřejmě vypustit některé body, ale musíme postupovat v souladu s myšlenkou modelu)

- většinou nelze používat paralelně více modelů, ale pro různé části projektu se mohou hodit různé modely

Zaměření a rozsah modelů

- velmi podrobné (definují všechny postupy v rámci vývojového cyklu) – definovány zpravidla většími firmami, ve kterých je vývoj softwaru důležitým prvkem činnosti (konzultační firmy, organizace vyvíjející interní software – banky, pojišťovny, apod.)

- méně podrobné (a také obecnější) – popisují celkovou koncepci vývoje, role pracovníků, postupy testování, apod.

- zaměřené na určitou fázi vývojového cyklu (sběr požadavků, testování apod.)
- existence vlastního firemního modelu – vnitrofiremní pravidla aplikovaná na obecnější model -> výhody:
  - nový pracovník si přečte dokumenty – na některé věci přijde rychleji – není třeba se mu to tolika věnovat
  - prevence před problémy sw vývoje – méně koncepčních chyb, případně jejich rychlejší a snadnější náprava
  - snadnější a úspornější komunikace v týmu – některé postupy jsou předem dány

Modely zaměřené na technickou část projektu

- důraz především na návrh aplikace
- obvykle popisují fázi kódování a testování
- chybí úvodní fáze projektu, sběr požadavků (nezávislá na technologii) – chybí natolik, že se ukazuje ->

Vývoj sw by měl být postaven tak, aby umožňoval co největší interakci se zákazníkem (důležitější než např. použitá technologie)

Tvorba modelů často doménou konzultačních firem (nedostatečná komunikace se zákazníkem měla u nich největší dopad) – model často bývá i tajný dokument

vs.

modely zveřejněné na internetu, knižně apod. (vytvořené na univerzitách či v open-source komunitách)

CASE nástroje

- softwarový produkt + konkrétní model pokrývající určitou fázi vývojového procesu (a proto jsou tyto nástroje drahé)

Prodej samotného modelu

- forma licence
- vývoj modelu stojí velké náklady (je to v podstatě intelektuálně velmi náročný sw produkt)
- komplexní modely (např. RUP – viz později)

Rizika a slabiny použití modelů

- Obecný model musí umožňovat více variant pro různé typy projektů (malé projekty vs. velké projekty – jiné nároky na formální řízení - vedoucí projektu se musí rozhodnout, kterou variantu a případně pro kterou část projektu použít – toto rozhodnutí je velmi důležité)

- Použití nesprávných, příliš náročných a složitých modelů – nižší efektivita vývojového týmu („utopení se“ v modelu)

- Všichni členové týmu musí modelu porozumět a akceptovat jej – je třeba čas k porozumění i k akceptaci

- Postupné zavádění a ukázky výhody vzniklých postupů

- Pozor na přehnaný důraz na dokument (dokument vytvořený jen kvůli naplnění modelu a nepotřebný pro projekt = ztráta času) – výsledek práce je funkční zdokumentovaný program, nikoli sada dokumentů

- Model je průvodce projektem (vede nás k cíli, tj. fungujícímu programu, méně snadno zabloudíme, někdy ho však můžeme slepě následovat a dostat se na scestí)

- Model sám o sobě nenahradí zkušené vývojáře, je to pouze pomocný nástroj

- Model nenahradí technologii

- Mnoho specialistů na modely nerozumí technologii – najímají si programátory – vzniklé produkty nepatří mezi nejkvalitnější (i když splňují zadání)

- Nutné přizpůsobení modelu vlastním potřebám (je však nutné vědět či dobře odhadnout, co ne/můžeme vynechat)

Modely a certifikáty

- normy ISO
- model – odborné rozhodnutí vs. certifikace – obchodní rozhodnutí
- obecně certifikace v oblasti vývoje softwaru dnes zřejmě přeceňována
- certifikace může znamenat jen formalizaci dosud používaného modelu

- různé typy certifikací
  - ne certifikace používané pro výrobní procesy (mají zaručit jejich opakovatelnost) – proces vývoje sw produktu opakovatelný není
  - ano certifikace zaměřené na projektově orientované organizace a činnosti – počítají s tím, že každý projekt představuje nový a neopakovatelný úkol, požaduje se, aby všechny projekty stejného typu měly stejnou strukturu činností
  - ano certifikace na dodržení kvality (v oblasti vývoje sw produktu především testování, údržba, tvorba dokumentace) – nelze však kvalitu zajistit pouhým dodržováním pravidel

„Pravěk“ softwarového inženýrství - 60. léta

- vývoj sw náročný, nejlepší specialisté, neuspokojivé výsledky
- první formalizované postupy, jak úspěšně tvořit software

- veškerá práce kolem počítačů (i využívání aplikací) – akademicky vzdělaní specialisté, vědci, ekonomové apod. – předpoklad podrobných teoretických znalostí
- předpoklad teoretických znalostí i u zadavatelů úkolů – přesné formální definice požadavků na začátku vývojového cyklu

- postupy složité, nepružné, odstraňování chyb v zadání nákladné
- vývoj softwaru v podstatě ve stejné kategorii jako výzkum – cíl si určoval programátor na základě vágního zadání (finance pravidelně přidělované zaměstnavatelem, časový termín vágní, požadavky na dotažení programu až na výjimky žádné)

- všeobecně uznávaný model – vodopád

Vodopádový model (waterfall model)

- výše uvedené 4 aktivity chápe jako samostatné fáze procesu
- princip vodopádu – vývojový cyklus rozdělen na několik přesně definovaných fází - po splnění se fáze ukončí a přechází se na další

Evoluční modely vývoje

- specifikace, vývoj a validace jsou smíšeny
- ze specifikace je velmi rychle vyvinut prvotní systém
- na základě požadavků zákazníka je systém dále upravován
- pak je systém předán, pro snazší údržbu pak může být i reimplementován

Formální modely vývoje (formální transformace)

- na začátku formální (matematická) specifikace systému, která je posléze transformována do funkčního programu
- transformace zachovávají správnost – na konci víme, že program odpovídá specifikaci

Sestavení systému ze znovupoužitelných (reusable) komponent

- předpokládá se, že části systému již existují
- vývoj systému se zaměřuje na integraci těchto částí

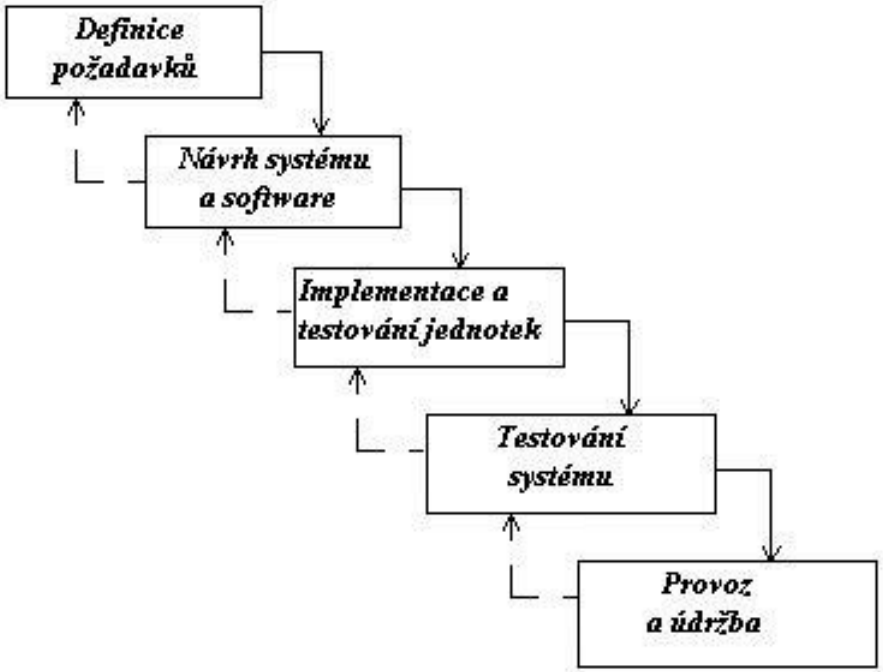
Iterativní modely – inkrementální vývoj a spirálové modely

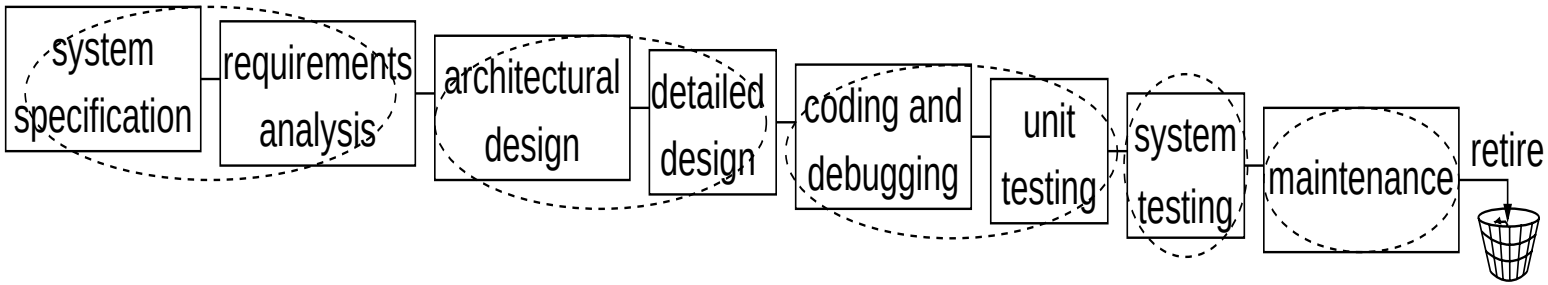
- hybridní modely – různé přístupy pro různé části systému (odpovídají potřebě tvorby velmi rozsáhlých systémů)

Neexistuje nejsprávnější model, různé modely jsou vhodné pro různé situace. V praxi snaha o kombinaci vodopádového modelu (nejlépe rozumově postihnutelné, organizovatelné) a evolučním a iterativním vývoji (důraz na potřeby zákazníka). Změna se dá očekávat ve vytváření softwaru z existujících komponent. Formální modely – jejich úspěšné použití pouze ve velmi specifickém prostředí.

Vodopádový model vývoje SW

- První model vývoje SW
- Schémata se mohou značně lišit v závislosti na literatuře viz následující dva obrázky (rozdělení do jednotlivých fází můžeme nazvat různě, principy ovšem zůstávají tytéž):




system specification
requirements analysis
architectural design
detailed design
coding and debugging
unit testing
system testing
maintenance
retire

Přehled základních aktivit:

Analýza domény, získávání a definice požadavků

- analýza domény – seznámení se s širším kontextem systému, oborem, či částí oboru, pro který je systém vytvářen,

- získávání (sběr) požadavků – konzultace s uživateli systému (na různých úrovních firemní hierarchie) – zjištění cíle, požadovaných služeb systému a omezení kladených na systém,

- definice požadavků – výše uvedené cíle, služby atd. jsou podrobně definovány v dokumentu specifikace požadavků (DSP), dokument slouží jako specifikace vytvářeného systému (grafické notace, např. UML)

Návrh systému a návrh SW (system&software design)

- určení celkové architektury systému, požadavky na HW a SW
- návrh SW – identifikace a popis základních abstrakcí a jejich vztahů (často grafická notace, např. UML)

Implementace a testování modulů

- realizace návrhu – množina modulů, tříd, programů…
- testování jednotek – ověření, že každý modul odpovídá specifikaci modulu

Integrace a testování systému

- jednotlivé moduly jsou sestaveny do výsledného systému
- úplný systém je otestován na shodu se specifikací
- po otestování je systém předán zákazníkovi

Provoz a údržba (maintenance)

- nejdelší fáze životního cyklu – systém se prakticky používá
- údržba = změny software vynucené provozem (nikoli administrace systému), tj. oprava chyb implementace (a také návrhu, či dokonce analýzy), které nebyly odhaleny v předchozích fázích + rozšiřování systému podle nových požadavků

Vodopádový model čistě teoreticky

- další fáze začne, když předchozí (úspěšně) skončí, **ale:** během návrhu se odhalí problémy s požadavky, během implementace problémy s návrhem, apod.

SW proces se nedá realizovat jako lineární sekvence

- musíme se vracet – (většinou o 1 fázi, ale…) – viz obrázek
- údržba způsobuje změny v určité fázi, pak zase vodopád

Veškeré návraty jsou pracné, drahé (zahrnují i přepracování dokumentů)

–> po několika iteracích dojde ke **zmrazení příslušné části vývoje** a přejde se na další fázi (Jak to odhadnout?... to chce zkušenost…) a to znamená, že:

- případné problémy se odkládají na později, čili jsou fakticky ignorovány,
- zmrazení požadavků často vede k tomu, že systém nedělá, co uživatel chce,

- zmrazení designu obvykle vede ke špatně strukturovaným systémům (problémy designu se pak při implementaci obcházejí různými triky, které ztíží další vývoj – tzv. „informační skleróza“)

Nedostatky a problémy základního modelu, aneb proč vodopádový model nefunguje, tak jak bychom chtěli (v průběhu let vedly k jeho modifikaci):

- Prodleva mezi zadáním projektu a vytvořením spustitelného systému je příliš dlouhá (zákazník může mít pocit, že se nic neděje)

- Rozdělení projektu do fázi je málo flexibilní

  - je obtížné reagovat na změny požadavků ze strany zákazníka - proto je vodopádový model vhodný pouze, pokud je problém dobře známý (výsledek závisí na úplném a korektním zadání požadavků, ale: uživatelé systému zřídka vědí, co přesně chtějí a co vědí, neumějí vyjádřit)
  - i kdybychom mohli vyjádřit všechny požadavky, na některé detaily přijdeme až ve fázi implementace,
  - i kdybychom znali všechny detaily, jako lidé neumíme s tak složitými věcmi nakládat,
  - i kdybychom s nimi uměli nakládat, vnější síly vedou ke změnám požadavků, některé z nich vedou ke zrušení platnosti dřívějších rozhodnutí,
  - problémy s vodopádovým modelem vedly k formulaci alternativních modelů

Klady vodopádového modelu:

- model je sice nedokonalý, ale je lepší mít nějaký model než nechat zvítězit naprostý chaos při řešení projektu
- první návod na vytvoření kostry postupu vývoje
- přehledný – použitelný pro malé projekty, či jako první přiblížení (hrubé zrno)
- výhodný z hlediska organizace práce dodavatele

Evoluční modely vývoje SW

základní myšlenky:

- vytvořit počáteční implementaci a vystavit jí komentářům uživatele (zákazník se dostává do centra pozornosti)
- počáteční implementaci postupně vylepšovat přes další meziverze tak dlouho, dokud nedostaneme adekvátní výsledek (vše na základě další interakce se zákazníkem a jeho dalších, nových, pozměněných atd. požadavků)
- tj. fáze specifikace, vývoje a validace neprobíhají vždy sekvenčně, ale mohou probíhat i paralelně, mezi fázemi je zpětná vazba
- modely vedené riziky

2 základní typy modelů evolučního vývoje SW:

1. model výzkumník (exploratory development, tj. průzkumný vývoj)
2. model prototyp (throw-away prototyping, tj. doslova tvorba prototypu, který bude zahozen)

Klady a výhody evolučního vývoje SW:

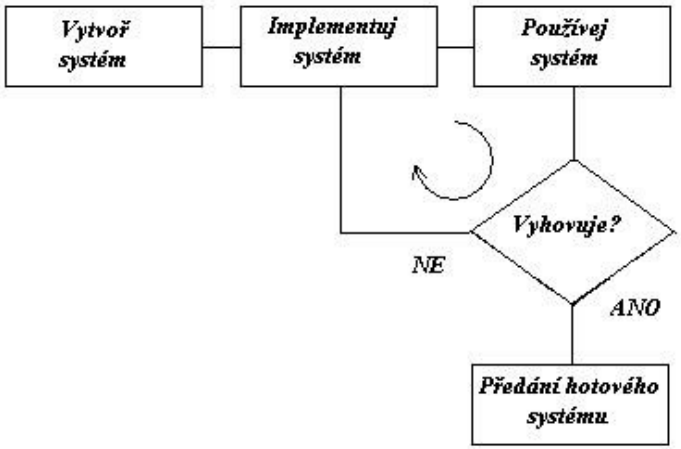
- vede efektivněji než vodopádový model k systému splňujícímu bezprostřední požadavky zákazníka
- specifikace může být vytvářena postupně
- zákazník vidí, že se něco děje

Problémy evolučního vývoje SW:

- proces není viditelný - manažeři nemohou měřit postup vývoje (na rozdíl např. od vodopádového modelu), problematická organizace z pohledu dodavatele
- systémy jsou často špatně strukturované (neustálé změny vedou k porušení struktury systému) -> vývoj dalších verzí se postupně stává obtížnější a dražší

Model výzkumník

- cílem je spolupracovat se zákazníkem na zjištění jeho požadavků,
- abychom zákazníkovi dodali požadovaný systém
  - vývoj obvykle začíná dobře srozumitelnými částmi systému
  - systém se vyvíjí přidáváním nových vlastností navrhovaných zákazníkem
- během vývoje systému se často vracíme k předchozím etapám - základní charakteristika vývoje.



Klady modelu výzkumník:

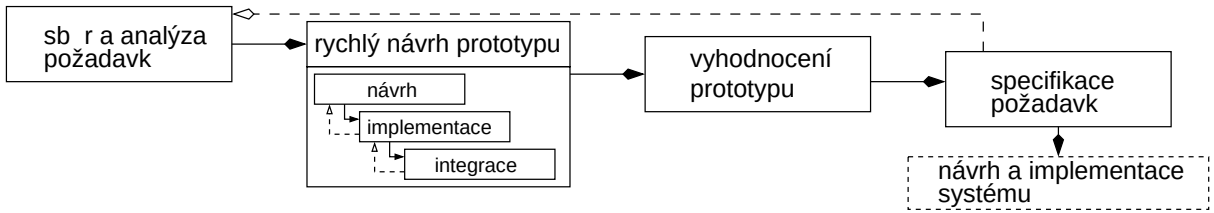
- systém je velice dobře přizpůsobitelný i dodatečným požadavkům zákazníka.

Problémy modelu výzkumník:

- jednotlivé etapy lze stěží plánovat časově, finančně i personálně
- dokumentace je většinou zhotovena po dokončení projektu (málokdy psána průběžně). Při údržbě pak není jasné, které požadavky jsou původní (byly stanoveny při zadání) a které jsou nové.

Model prototyp

- cílem je lepší pochopení zákazníkových požadavků a v důsledku vytvoření lepší definice požadavků na systém (opět důraz na zákazníka)
- tvorba prototypu se zaměřuje na ty části požadavků zákazníka, kterým moc nerozumíme
- prototyp je částečně zavedený produkt se všemi vnějšími rozhraní



Klady modelu prototyp:

- prototyp je testován a ověřován budoucími uživateli -> je upřesňována specifikace požadavků.

Problémy modelu prototyp:

- náročný na vedení v případě existence třeba i několika nezávislých vývojových skupin (zvýšené náklady projektu)
- může vyžadovat speciální nástroje a techniky -> problematický výstup, např. nekompatibilní s jinými nástroji a technikami

Evoluční vývoj SW vhodný pro:

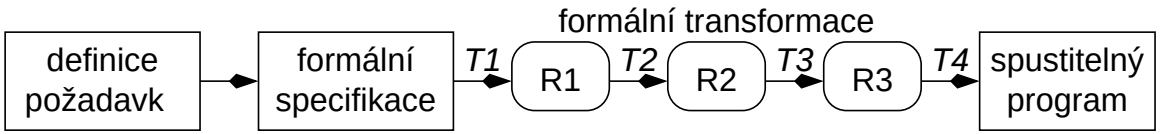
- malé systémy (do 100 000 řádků kódu),
- středně velké systémy (do 500 000 řádků) s krátkou dobou života

Ve velkých systémech by evoluční vývoj působil zřetelné problémy, varianta smíšeného přístupu:

- špatně srozumitelné části - throw-away prototyping => zpřesnění DSP
- dobře srozumitelné části - vodopádový model
- uživatelské rozhraní - model výzkumník

Formální modely vývoje SW

- určitým způsobem podobné vodopádovému modelu
- na počátku je vytvořena specifikace požadavků
- ta je zpřesněna do podrobné formální specifikace systému (pojmem formální se zde míní "matematická")
- postupné zpřesňování specifikace formálními transformacemi, konečným výsledkem spustitelný program



- v každém kroku se přidávají podrobnosti, ale můžeme dokázat, že odpovídá reprezentaci systému z předchozího kroku
- pokud neuděláme chybu, pak výsledek (dokazatelně) odpovídá specifikaci
- proces designu, implementace a testování jednotek je nahrazen formálními transformacemi
- dokázat korespondenci transformací je obtížné a lze při tom udělat chybu

- proto jsou v praxi mnohem oblíbenější formální metody, pro které existuje podpůrný SW - model checkers, theorem provers ("dokazovače hypotéz")

Klady formálního modelu:

- zejména tam, kde je třeba dokázat zákazníkovi bezpečnost a spolehlivost systému
- cena vývoje srovnatelná s jinými přístupy

Problémy formálního modelu:

- nepoužitelný pro klasické problémy a klasického zákazníka
- vyžaduje specializované znalosti - výběr transformace vyžaduje zkušenosti lišící se od klasického programování
- někdy vyžaduje transformaci do speciálního programovacího jazyka (často čím nižší programovací jazyk, tím obtížnější převod)
- z uvedených důvodů se v praxi příliš nepoužívá

Iterativní modely vývoje SW

Základní myšlenky:

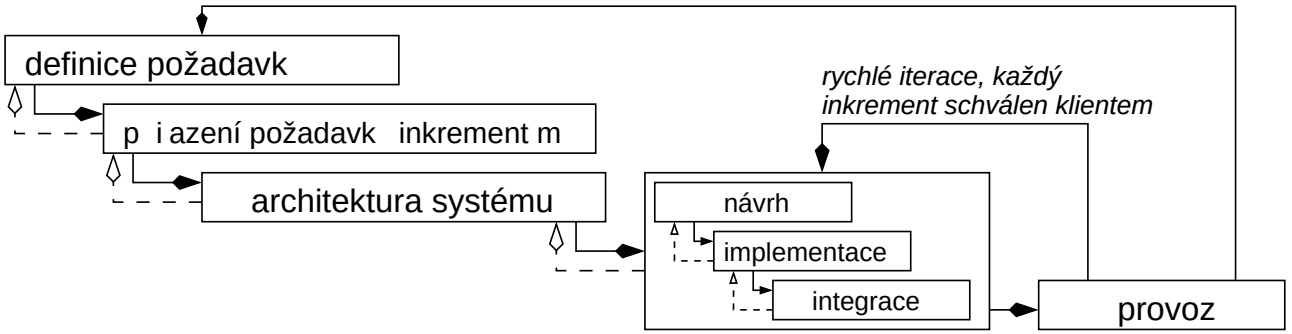
- počítají s vývojem požadavků v čase (opět více zaměřeno na zákazníka než vodopádový model) => nutnost několika (i několika mnoho) iterací procesu vývoje
- podporují myšlenku, že velké systémy je možné rozdělit na menší, je možné používat různé přístupy pro různé části, tzv. hybridní modely

2 modely navržené speciálně pro iterativní procesy vývoje:

- inkrementální vývoj SW (Lehman 1969, Mills 1980)
- spirálový model vývoje (Boehm 1988) a WinWin spirálový model (Boehm a Bose, 1994)

Model inkrementálního vývoje

- česky často nazýván "přírůstkový model"
- navržen jako způsob, jak omezit přepracovávání částí v důsledku změn požadavků (inkrement = malý přídavek)
- někdy umožňuje zákazníkovi odložit rozhodnutí o detailních požadavcích, dokud nezíská zkušenost se systémem



Průběh procesu:

- definují se přehledové požadavky (někdy pouze cíl projektu)
- požadavky se přiřadí jednotlivým inkrementům
- navrhne se architektura systému
- opakovaně pro každý inkrement - dokud nevytvoříme finální systém: vyvineme a validujeme (otestujeme) inkrement, integrujeme inkrement a validujeme systém.

Model přepokládá, že:

- na začátku zákazník identifikuje přehledově služby, které od systému požaduje,
- zákazník určí priority, tj. které služby jsou pro něj nejdůležitější a které jsou nejméně důležité
- pak je definována množina inkrementů, každý inkrement poskytuje podmnožinu celkové funkčnosti (alokace služeb do inkrementů závisí na prioritě služby, nejdůležitější služby mají být zákazníkovi předány jako první)
- v dalším kroku jsou detailně specifikovány požadavky na první inkrement
- inkrement je vytvořen nejvhodnějším procesem vývoje; během vývoje nejsou akceptovány změny požadavků na vytvářený inkrement
- spolu s prvním inkrementem mohou být implementovány společné služby systému (občas lze společné služby vytvářet také inkrementálně)
- paralelně s vývojem může probíhat analýza požadavků pro další inkrementy
- po dokončení inkrementu může být tento předán zákazníkovi, který ho může používat - získá tím část funkčnosti systému
- zákazník může se systémem experimentovat, což mu pomůže upřesnit požadavky na další inkrementy nebo na novější verze předaného inkrementu

Klady a výhody inkrementálního přístupu:

- zákazníci nemusejí čekat na dokončení celého systému, aby z něj něco měli
- počáteční inkrementy jim mohou pomoci získat zkušenost pro definici dalších inkrementů
- je menší riziko selhání celého projektu; i když některé inkrementy mohou být problematické, je pravděpodobné, že většina jich bude úspěšně předána zákazníkovi
- protože zákazníkovi předáváme jako první inkrementy s nejvyšší prioritou, budou nejdůležitější části systému nejlépe otestovány; jinými slovy - zákazníka pravděpodobně nepotká havárie nejdůležitější části systému

Problémy inkrementálního modelu:

- inkrementy by měly být relativně malé (do 20 000 řádek kódu),
- na druhou stranu by každý inkrement měl poskytovat nějakou navenek viditelnou funkčnost - proto může být obtížné namapovat požadavky zákazníka na inkrementy správné velikosti
- problematický inkrement může zpětně ovlivnit již hotovou část systému (při detailnější specifikaci požadavků inkrementu se zjistí, že se předtím na něco zapomnělo), oprava nemusí být triviální
- většina systémů obsahuje množinu základních služeb, které jsou vylaďovány různými částmi systému, v inkrementálním modelu vývoje se ale požadavky nespecifikují detailně, dokud se nedostaneme k jejich vývoji => je obtížné identifikovat služby potřebné pro všechny inkrementy - proto je pro větší projekty vhodné doplnit inkrementální model o fázi analýzy, podobnou jako ve vodopádovém modelu

Agilní metodiky (podrobněji ASWI)

- metodiky odpovídající inkrementálnímu modelu

Extrémní programování (Beck 1999)

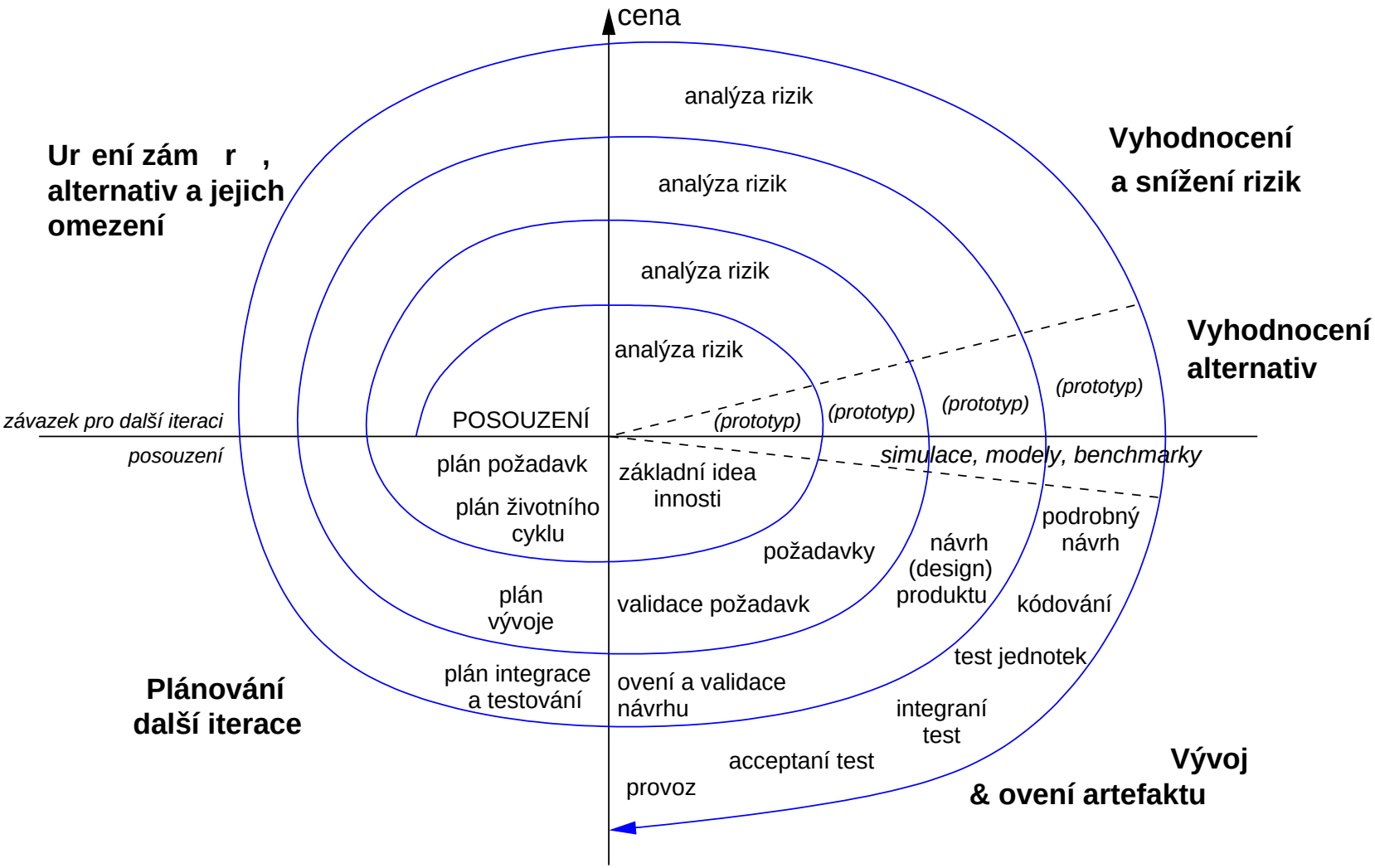
- rozporuplné názory
- krátký průzkum, případy užití,
- vývoj malých inkrementů,
- zatažení zákazníka do procesu,
- průběžné vylepšování kódu,
- pro případy, kdy požadavky na sw jsou vágní nebo se rychle mění,
- použitelná pro malé týmy (2-10lidí)
- veškerý kód psán povinně ve dvojicích (jeden kóduje, druhý kontroluje)

Další agilní metodiky např. DSDM, Scrum, http://www.agilealliance.org

Příklad SW procesu pracujícího podle inkrementálního modelu - Rational Unified Process (**RUP**), popis najdete na **http://www.kiv.zcu.cz/studies/publikace/** (ASWI)

Spirálový model SW procesu

- spirálový model SW procesu původně navrhl Boehm (1988)
- realistický přístup k vývoji rozsáhlých SW systémů, orientován na rizika, místo sekvence (kde se chtě nechtě někdy musíme vracet) je proces
- reprezentován spirálou, v každé otočce vytváříme úplnější verzi SW
- SW projekt rozděluje do miniprojektů
  - každý miniprojekt řeší jedno nebo více rizik, dokud nejsou všechna hlavní rizika vyřešena
  - pojem "riziko" je definován poměrně široce, může znamenat špatně pochopené požadavky, nesrozumitelnou architekturu, potenciální problémy s výkonností, potíže s technologií apod.
  - po vyřešení rizik pokračujeme v miniprojektu, např. podle vodopádového modelu


cena
analýza rizik
analýza rizik
analýza rizik
analýza rizik
Ur ení zám r ,
alternativ a jejich
omezení
Vyhodnocení
a snížení rizik
Vyhodnocení
alternativ
závazek pro další iteraci
posouzení
POSOUZENÍ
(prototyp)
(prototyp)
(prototyp)
(prototyp)
simulace, modely, benchmarky
plán požadavk
základní idea
innosti
plán životního
cyklu
požadavky
návrh
(design)
produktu
podrobný
návrh
plán
vývoje
validace požadavk
kódování
test jednotek
plán integrace
a testování
ovení a validace
návrhu
integraní
test
Plánování
další iterace
acceptaní test
provoz
Vývoj
& ovení artefaktu

*

Spirálový model začíná v malém ve středu spirály, prozkoumáme rizika a pak zvolíme způsob řešení další iterace

Každá otočka spirály je rozdělena do 4 sektorů (sektory nemusejí trvat stejnou dobu):

- určení záměrů pro současnou fázi projektu (= otočku spirály)
  - definujeme záměry dané fáze projektu (funkčnost, výkonnost apod.)
  - určíme alternativní možnosti realizace záměrů (např. design A, design B, použití existující komponenty, nákup komponent)
  - identifikujeme omezení daných alternativ (cena, časový plán, přenositelnost)
  - vyhodnotíme alternativy vzhledem k záměrům a omezením - často se tím ukáže nějaká nejistota, která je zdrojem rizik
  - riziko = něco co se může nepovést
  - pokud vyplynula rizika, měli bychom se jimi zabývat v další fázi

- vyhodnocení a snížení rizik
  - pro každé z identifikovaných rizik je provedena podrobná analýza a kroky ke snížení rizika

- to může znamenat prototypování, simulace, vypracování dotazníků apod. (např. riziko, že jsou nesprávně definovány požadavky, můžeme vytvořit prototyp části systému a konzultovat ho se zákazníkem)

- vývoj a validace
  - provádíme vývoj a ověření produktu další úrovně
  - použitý model SW procesu je určen zbývajícími riziky (riziko integrace podsystémů -> vodopádový model, riziko nevhodné uživatelské rozhraní -> evoluční prototypování)

- plánování
  - naplánujeme další fázi projektu (organizace, požadavky na zdroje, rozdělení práce, časový plán, výstupy)
  - každá otočka je zakončena posouzením všech produktů vytvořených v předchozím cyklu, plánů pro další cyklus a potřebných zdroj
  - cílem je aby byly všechny zúčastněné strany srozuměny s plány na další fázi

Obrázek - příklad, jak může vypadat vývoj podle spirálového modelu:

- nejvnitřnější otočka spirály - všeobecný záměr, studie proveditelnosti
- druhá - definice požadavků
- třetí - návrh (design) systému
- čtvrtá - vytvoření všech programů

uvedený diagram je pouze příklad, konkrétní počet iterací nebo kroky v iteracích se mohou lišit

Spirálový model je chápán jako "generátor modelů SW procesu", s jeho pomocí vytváříme podrobnější model SW procesu (tj. konkrétní obsah jednotlivých otoček může být odlišný)

Rozdíly oproti předchozím modelům:

- explicitně uvažuje rizika projektu - hlavní problém modelu - je závislý na zkušenosti vývojářů najít a realisticky posoudit zdroje rizik (pokud některé riziko podcení, nevyhnutelně nastanou potíže)
- v modelu nenajdeme pevné fáze jako je specifikace nebo návrh – model je obecnější, může různým způsobem zahrnovat ostatní modely SW procesu:
- model je aplikovatelný i na jiné typy projektů, než je vývoj SW

- model široce známý a oblíbený, je ale méně používán než klasický vodopádový model (zákazníci jsou zvyklí na vodopádový model, je to předpoklad mnoha zadání)

WinWin spirálový model

původní spirálový model neposkytuje podrobnosti o tom, co se má dělat v prvním kvadrantu (neříká, odkud se vezmou "záměry pro současnou fázi projektu") -> modifikován (Boehm a Bose, 1994)

nový model je postaven na tzv. win-win přístupu: snaha, aby všechny zúčastněné strany (zákazníci, vývojáři, vedoucí atd.) byly "výherci"

snaha vyrovnat se s konfliktními zájmy

- zákazník: mít produkt co nejdříve, zaplatit za něj co nejméně
- vývojáři: odborný růst, pracovní a platové podmínky, preference pro používané nástroje (chci to psát v Javě), odložit psaní dokumentace apod.
- nadřízení: zisk, nepřekročení rozpočtu, žádná překvapení

Postup

1. identifikace klíčových hráčů - do úvah musíme zahrnout všechny hráče podstatné pro daný cyklus projektu (např. nadřízené uživatelů apod.)
2. pochopení zájmů každého hráče (motivační analýza, obvykle nejobtížnější část - motivace jiných lidí budou totiž odlišné od našich)
3. urovnání podmínek pro výhru, řešení konfliktů - např. zákazníci většinou neodhadnou, co je snadné nebo obtížné naprogramovat => mají nerealistická očekávání - nabídnout novou alternativu typu výhra-výhra

Ostatní části jsou stejné jako u původního spirálového modelu.

Vývoj orientovaný na znovupoužitelné komponenty

- vývojáři znají kód nebo návrh podobný tomu, který se požaduje, ten vezmou, upraví a začlení do svého systému - děje se nezávisle na modelu vývoje SW

Component-based SWE

- spoléhá na existenci velké množiny SW komponent a rámce pro jejich integraci
- komponentou např. knihovní fce, třída, podsystém, aplikace
- rámcem např. příkazové jazyky (Tcl/tk), distribuované objektové architektury (CORBA, JavaBeans) apod.

Komponentově orientovaný proces vývoje zahrnuje:

- specifikaci požadavků
- **analýzu komponent**
- **modifikaci požadavků**
- návrh systému s využitím komponent
- vývoj a integraci
- validaci systému

Analýza komponent

- nejprve vyhledáváme komponenty, které odpovídají specifikaci požadavků
- nalezené komponenty obvykle specifikaci nesplňují přesně, resp. poskytují pouze část požadovaných funkcí
- výsledkem je informace o dostupných komponentách, jejich funkčnosti apod.

Modifikace požadavků

- provedeme analýzu a modifikaci požadavků tak, aby požadavky odrážely dostupné komponenty
- pokud nemůžeme změnit požadavek, vrátíme se k analýze komponent se snahou najít alternativní řešení

Návrh systému s využitím komponent

- během této fáze je navržen rámec systému nebo se využije už existující rámec
- pokud některá potřebná komponenta neexistuje, musíme jí navrhnout

Vývoj a integrace

- vytvoříme komponenty, které nemáme (a nemůžeme koupit atd.)
- komponenty integrujeme do systému

Výhody a klady modelu:

- omezené množství SW, který musíme sami vyvinout => snížení ceny produktu a menší nebezpečí, že nastanou potíže
- obvykle rychleji získáme výsledný SW

Nevýhody modelu:

- kompromisy vůči požadavkům jsou nevyhnutelné => systém nemusí splňovat požadavky zákazníka
- vytvářené části musíme přizpůsobit možnostem komponent, což se projeví na kvalitě jejich návrhu (jejich flexibilitě apod.)
- vývoj není zcela v našich rukách, protože nové verze komponent vyvíjí někdo jiný (vývoj komponenty může být ukončen, nové verze nemusejí být kompatibilní se starými verzemi) => údržba systému se může prodražit
- rámce a knihovny komponent bývají tak rozsáhlé, že seznámení s nimi může vyžadovat několik měsíců (ve velkých organizacích na to bývají mají vyčleněné konkrétní lidi)

Shrnutí

vodopádový model - základní model, konzistentní se strukturovaným programováním shora dolů; je vhodný, pokud jsou známé požadavky (např.: operační systémy, překladače apod.)

- evoluční vývoj - pokud části požadavků nejsou zřejmé, např. uživatelské rozhraní ("nevím co chci, ale poznám to, až to uvidím")
- formální metody - pokud musím dokazatelně splnit specifikaci, mám podpůrné nástroje a tým seznámený s formálními metodami
- komponentově orientovaný vývoj - máme-li vhodné komponenty
- inkrementální vývoj - potřebujeme omezit přepracovávání, dodáváme systém po částech
- spirálový model - vhodné pro složité inovativní projekty vytvářené uvnitř organizace, zahrnuje předchozí modely jako speciální případy (projekt nemusí nemusí být SW, ale např. i HW)
- win-win spirálový model - jako spirálový, ale vhodnější pro projekty na zakázku.

Hlavním účelem modelů je určit správné pořadí kroků při vývoji SW a určit kritéria pro přechod k dalšímu kroku

Model SW procesu odpovídá na otázky:

- co budeme dělat příště?
- jak dlouho to máme dělat?