1a. Modely životního cyklu SW. Návaznosti a produkty jednotlivých etap. Aplikace CASE v životním cyklu. Specifikace požadavků. Prototypy a oponentury.

Osnova

1. Modely životního cyklu SW
2. Návaznosti a produkty jednotlivých etap
3. Aplikace CASE v životním cyklu
4. Specifikace požadavků
5. Prototypy a oponentury

Výklad

1. Modely životního cyklu SW

Životní cyklus SW popisuje život SW od jeho návrhu, přes implementaci, až po jeho předání a údržbu. Je to síť koroků potřebných k vytvoření SW systému, obdoba byznys procesu.

Jednotlivé etapy:

Specifikace cílů, vize, plánování → Specifikace požadavků → Návrh → Implementace → Testování → Evaluace → Předání a nasazení → Provoz a údržba

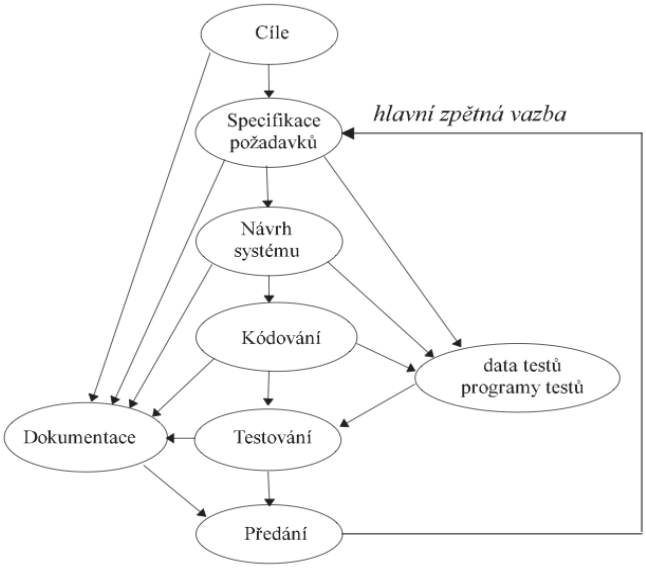
1. Specifikace cílů – proč a taky trochu co
2. Specifikace požadavků – přesně co a taky trochu jak, studie proveditelnosti (feasibility study), zdroje, předběžná cena, termín
3. Návrh systému – technické otázky (dekompozice, rozhraní, struktura dat, algoritmy, SW nástroje, …)
4. Programování
5. Testování – části (unit testy), integrační, funkce, předávací
6. Oživení a předání – instalace, předávací (akceptační) testování, zkušební provoz
7. Udržba – odstraňování chyb objevených za provozu, úpravy/přizpůsobování, vylepšování funkcí
8. (Ztažení z provozu)

Jednotlivé typy:

Vodopád

- jeden z nejstarších modelů vývoje SW
- po úplném dokončení jedné etapy je její výsledek předán jako vstup pro další etapu – k zakončené etapě není nutné se vracet
- začnu-li padat, nezastavím se dříve, než se rozbiji o kámen zvaný předvedení
- problémy s cenou v případě nezdaru v některé z pozdějších etap

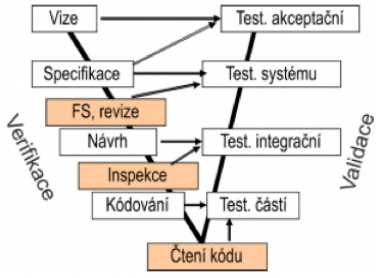
- existují však variace nebo vylepšení tohto modelu



Výhody	Nevýhody	Použití
• jednoduchý k porozumění a použití • poskytuje jasnou strukturu pro méně zkušený tým	• požadavky na systém musí být dopředu známé • málo flexibilní • může dávat mylný dojem progresu • chyby, ke kterým je potřeba se vracet zvyšují náklady • slabá kontrola uživatelem → systém uvidí až na konci, což může být pozdě • **nedostatky se projeví až na konci**	• požadavky na systém jsou dobře známé • definice produktu je stabilní • dokonalé pochopení technologií • nová verze existujícího produktu

V-shaped model

- upravená varianta vodopádu, která zdůrazňuje verifikaci (studie proveditelnosti, revize, inspekce,...)
- testování produktu je plánováno paralelně s jednotlivými fázemi vývoje
- jenom validace nestačí
- účinnost odhalení chyb se zvyšuje verifikací – může být až 80%

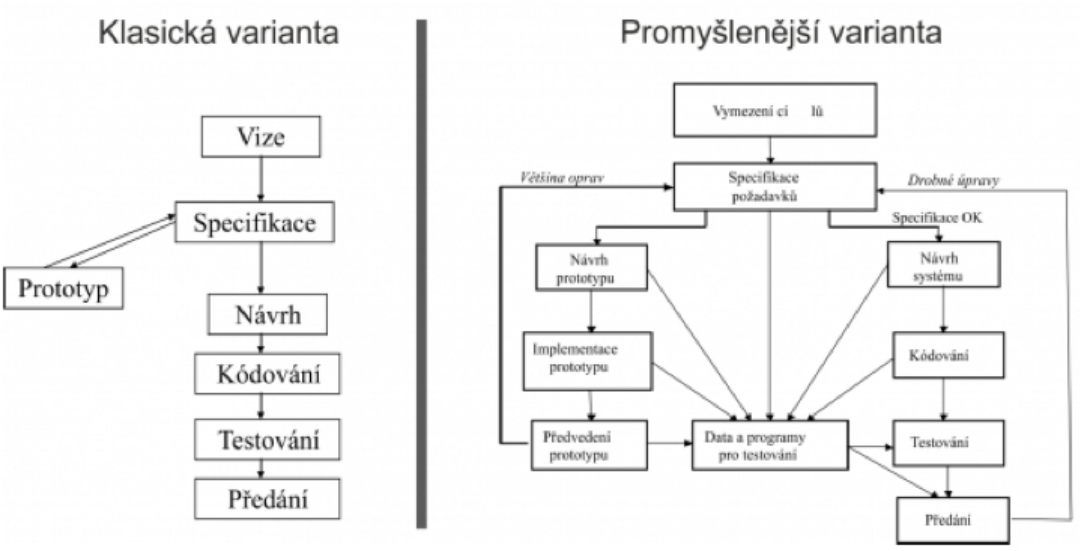


Výhody	Nevýhody	Použití
• jasné a jednoduché použití • důraz na naplánovaní verifikace a validace je kladený už na začátku vývoje • každý výstup z fáze musí být prověřený • projektový manažment může sledovat progres po jednotlivých milnících	• souběžné události se nedělají lehko • dynamické změny v požadavcích se nedělají lehko • (neobsahuje analýzy rizik) [**?**]	• pro vývoj systému, který vyžaduje vyšší spolehlivost • požadavky na systém jsou dobře známé • řešení a technologie jsou dobře známé

Prototypování

Prototyp je částečně funkční model systému, který realizuje nebo simuluje některé vlastnosti systému.

- vývojáři vytvoří prototyp v průběhu fáze specifikace požadavků (analýzy)
- tvorba prototypů probíhá za účelem získávání poznatků
- prototyp se prezentuje koncovým uživatelům, dává zpětnou vazbu vývojářům
- prototypy se pak případně vylepšují, použijí nebo zahodí

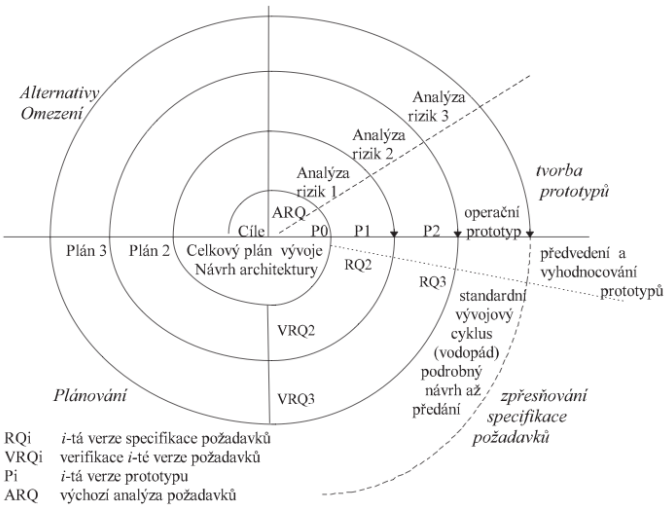


Výhody	Nevýhody	Použití
• zákazník „vidí“ požadavky na systém • vývojáři se učí od zákazníků (komunikace nad doménou) • více se upřesňuje	• tendence sklouznout k vývoji “code-and-fix” • špatná reputace za “quick-and-dirty” methods • je nutné stanovit hranici pro vytváření prototypů,	• spíše pro menší systémy • když jsou nepřesně formulovány, případně měnící se požadavky na systém, nebo požadavky musí být objasněny

koncový produkt • umožňuje větší flexibilitu návrhu a vývoje • protypy stimulují uvědomování si potřebné přídavné funkcionality	aby se nevytvářely donekonečna (scope creep)	• vývoj uživatelského rozhraní • pro případ rychlé demonstrace systému • nový, originální návrh • vhodné pro OO analýzu a návrh

Spirálový model

- vytvořený hlavně za účelem minimalizace rizik při postupu pomocí vodopádu
- jednotlivé kroky se ve spirále opakují se stále vyšším a vyšším stupněm zvládnutí problematiky
- stanoví se cíle a výchozí analýza požadavků a návrh architektury
- výchozí prototyp a jeho předvedení → plán vývoje prototypu → provede se analýza alternativ a rizik
- několikanásobně se provede životní cyklus prototypu
- poslední prototyp se nazývá operační → po něm následuje zbytek vodopádu (návrh, kódování, testování, předání, údržba)



Návaznost činností se získá procházením spirály ze středu ve směru hodinových ručiček.

Výhody	Nevýhody	Použití
• zahrnuje analýzu rizik • uživatel (zákazník) brzo vidí systém v podobe prototypu	• čas strávený nad vyhodnocováním rizik u projektů s malými nebo nízkymi riziky	• pro vývoj od počátku • vhodné pro velké systémy

• hodně rizikové funkce se vytvářejí jako první návrh • nemusí být úplně perfektní • zákazníci jsou lépe provázaní s každou fází životního cyklu • včasný a častý feedback od zákazníků • lepší odhad kumulativních nákladů	• celkově může být hodně náročné na čas • spirála by mohla končit v nekonečně :) • může být obtížné definovat konkrétní a ověřitelné mílníky	• vhodné pro IS, kde je známá míra nejistoty ve stanovení požadavků (uživatelé jsi nejsou jistí potřebami) • když je vyhodnocení nákladů a rizik důležité • pro středně až vysoko rizikové projekty

Hlavní odlišnosti od vodopádu:

- součástí každého cyklu je analýza rizik
- v každé iteraci se ověřují požadavky
- operační prototyp obsahuje vše potřebné

Výzkumník

Je to experimentování, u kterého často netušíme, jak dopadne v průběhu vývoje se řešitelé při získávání poznatků a zkušeností často vracejí k již přežitým etapám.

Charakteristika: Analyzuj a navrhni systém → Implementuj systém → Používej systém → Pokud vyhovuje, předej systém, pokud ne vrať se k nové implementaci

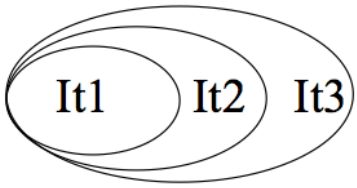
Nevýhody	**Použití**
• práce se obtížně řídí a plánuje • často dochází k překročení stanovených finančních i časových limitů • má problémy s dokumentací (neexistující či neplatná) • zpravidla do vývoje vidí jen jeden výzkumník nebo výzkumný tým a jejich rozpad či odchod pracovníka znamená ukončení projektu (nenahraditelnost řešitelů)	• tento model je vhodný řekněme pro programování věcí, které řešitelé neznají, pro experimentální projekty

Iterativní vývoj

Iterativně znamená „předělávat“. Iterační model se od spirálového modelu liší tím, že výsledkem každého cyklu je stále větší a lépe fungující část cílového systému. Prototypy se realizují pouze u těch požadavků, které jsou buď sporné nebo spojené s nějakými

významnými riziky. Iterační vývoj lze přirovnat k neustále přestavovanému domu, ve kterém se přistavují nová patra, což si vynucuje úpravy již existujících částí domu.

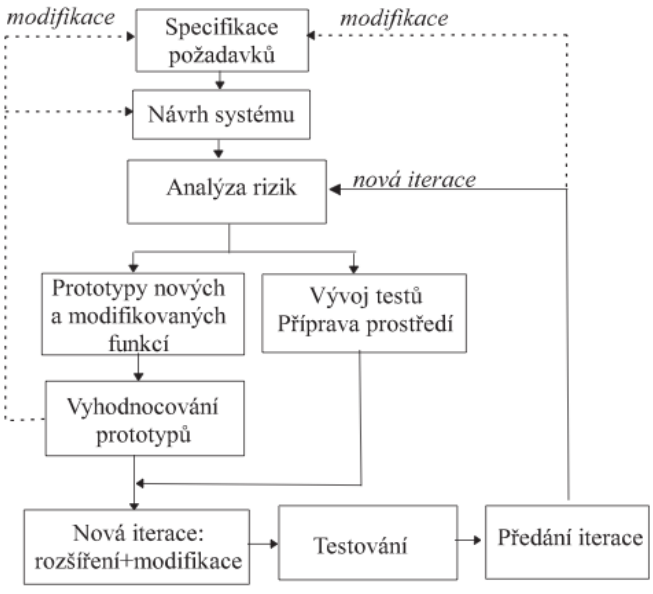
→ Jedna rostoucí aplikace. Pro vývoj iterace je potřeba mít hotové jádro. Typické pro agilní vývoj.

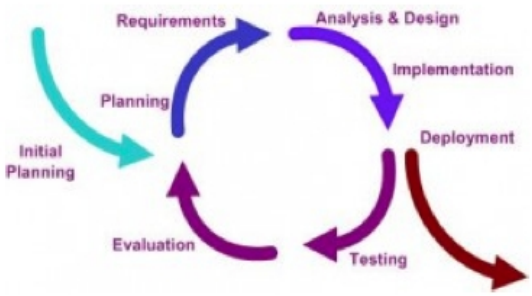


Obvyklé rozhraní: RPC, API

Vlastnosti:

- celý projekt se vyvíjí v několika iteracích
- iterace směřují k postupnému vylepšení, zpřesnění, dodělání nebo opravení části systému
- výsledkem každého cyklu je větší a lépe fungující část cílového systému
- každá iterace obsahuje analýzu, návrh, testování apod. (tj. miniaturní vodopád),ale s různou intenzitou, např.:
  - v první iteraci provést celkovou analýzu požadavků a obrysový plán vývoje, více rozpracovat jádro systému, implementovat základní testovací třídy
  - v druhé iteraci podrobně rozpracovat důležité části systému, rozmodelovat je a částečně implementovat
  - ve třetí iteraci podrobně rozpracovat méně podstatné části systému, doimplementovat věci z předchozí iterace
- vývoj typu vodopád je tedy iterativní vývoj s jedinou iterací



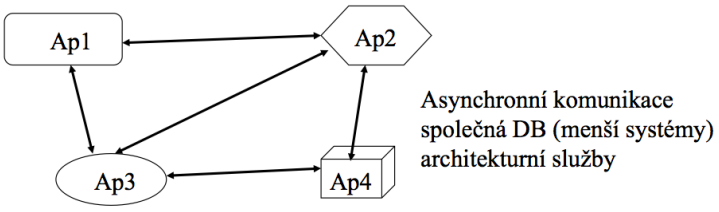


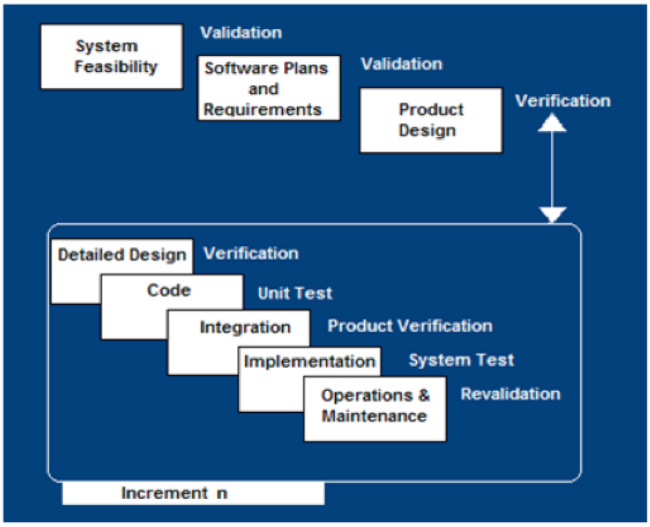
Výhody	Nevýhody
• rychlejší (dílčí) výsledky (umožňuje dříve dospět k SW (po částech)) • rychlejší odhalení chyb • lze vyvíjet i v menším týmu • snadněji se integrují produkty třetích stran a existující aplikace	• celý systém je hotový až za delší dobu • obtížně se zkracují termíny realizace • u některých systémů je obtížné použití • nutnost předělávat kód (není tak velká nevýhoda – je lepší špatný kód přepsat, než ho nějak obcházet)

Inkrementální vývoj

Inkrementálně znamená „přidávat k“. Podobá se iteračnímu vývoji. Každý přírustek znamená nový vývojový cyklus. Po dokončení se integruje do stávajícího systému. Přírustky jsou více méně samostatné systémy. Lze použít pokud je možné jednotlivé části dekomponovat do samostatných problémů.

→ Propojují se různorodé aplikace. Komunikace přes middleware, přístup ke společným datům (např. DB), asynchronní komunikace. Pro vývoj přírustku nepotřebují mít zbytek systému. Agilita potřebuje doladit.





Vlastnosti

- uplatňuje se zejména u větších projektů a/nebo v agilním vývoji
- jednotlivé části systému (přírůstky, inkrementy) vytváříme „nezávisle“ na zbytku a pak integrujeme
- vývoj jednotlivých přírůstků může probíhat iterativně, vodopádem, XP,... – nejčastěji

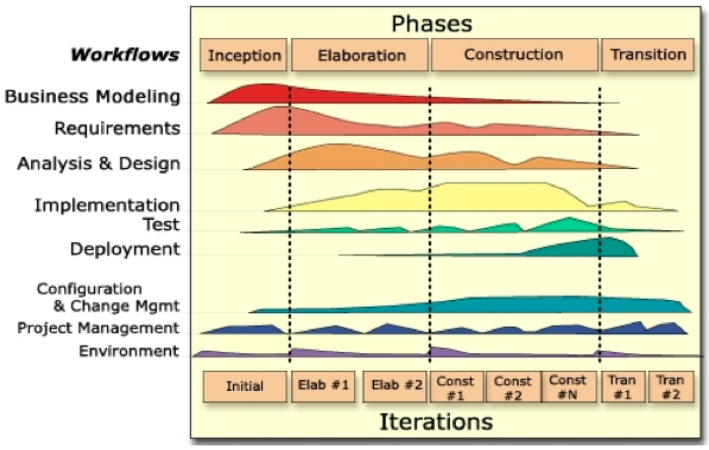
se používá iterativní vývoj přírůstků

- vývoj typu vodopád je tedy inkrementální vývoj s jediným přírůstkem představujícím celý systém.

Výhody	Nevýhody	Použití
• ve více týmech (možný paralelní vývoj) • integrace existujících aplikací a aplikací třetích stran • samotné přírustky lze realizovat různymi metodami • snadno lze modifikovat, modernizovat	• vyžaduje dobré plánování a návrh • vyžaduje dřívější definici kompletního a plně funkčního systému, aby se definovaly inkrementy • vyžaduje dobře zadefinovat rozhraní modulů	• projekty, které mají dlouhý plán vývoje • iterativní a inkrementální vývoj používejte pouze pro projekty, se kterými chcete uspět

RUP

- není jeden konkrétní ustálený proces, ale spíše framework určený k tomu, aby ho organizace implementovala a upravila dle svých potřeb a specifik
- aplikuje iterativně - inkrementální vývoj
- výrobek prochází přes čtyři základní fáze (inkrementy):
  - Zahájení (Inception): potvrzení konceptu, proveditelnost, cíle
  - Rozpracování (Elaboration): detailní požadavky, scénáře použití, architektura, komponenty, vzory interakce na vysoké úrovni
  - Konstrukce (Construction): zdokonalení architektury, implementační iterace řízené rizikem
  - Předání (Transition): uživatelské přijetí, ...
- každá fáze může být dále rozdělena do různých iterací (cyklů)
- každá iterace (cyklus) obsahuje analýzu, návrh, implementační aktivity a produkuje novou verzi systému – release (inkrement)

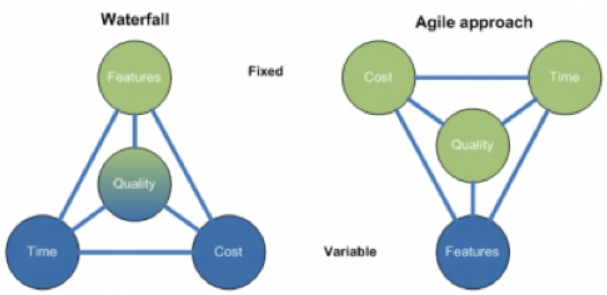


Agilní metodiky

Manifesto of Agile Software Development http://agileManifesto.org

Vlastnosti:

- metody zaměřené více na lidi – určujícím faktorem v úspěchu projektu je kvalita lidí pracujících na projektu a jejich spolupráce
- často iterativní nebo inkrementální vývoj, krátký vývojový cyklus (měsíc, 14 dní)
- malé ale výkonné týmy, střídání rolí
- nevhodné pro kritický SW, velké systémy, při neochotě zákazníka spolupracovat za běhu a pokud nemáme k dispozici kvalitní řešitele
- malý důraz na dokumentaci, program by měl být samodokumentující
- snaží se o dodržení rozpočtu a harmonogramu tím, že umožňuje změny funkcionality
- zapojení zákazníka do vývoje (zákazník se účastní sestavování návrhu a testů, ideálně je součástí vývojového týmu)
- individuality a interakce mají přednost před nástroji a procesy
- fungující software má přednost před obsáhlou dokumentací
- spolupráce se zákazníkem má přednost před sjednáváním smluv
- reakce na změnu má přednost před plněním plánu



Konkrétní metodiky:

- Adaptive Software Development (ASD)
- Feature Driven Development (FDD)
- Crystal Clear
- Dynamic Software Development Method (DSDM)
- Rapid Application Development (RAD)
- Scrum
  - sprinty (2-3 týdny); vstupem je backlog, který má na starost product owner (PO), ten je rovněž zodpovědný za výsledný produkt; scrum master (SM) zařízuje hladký průbeh sprintu; team (programátoři, testeři, technical writer, …); na začátku sprintu je planning kde se naplánují jednotlivé položky (user stories) z backlogu, včetně odhadů (story points); sprint zakončuje review (předvedení hotové funkcionality) a retrospektiva (hodnocení průběhu sprintu)
- Extreme Programming (XP) http://www.extremeprogramming.org
  - Extrémní protože myšlenky jsou dotaženy do extrémů. Tato metodika vychází z důrazu na čtyři základní hodnoty: komunikace, jednoduchost, zpětná vazba, odvaha a respekt. Patří sem např. párové programování.
- Test-Driven Development (TDD)
  - Prvním krokem je definice funkcionality a následné napsání testu, který tuto funkcionalitu ověřuje. Poté přichází na řadu psaní kódu a nakonec úprava tohoto kódu.

2. Návaznosti a produkty jednotlivých etap

Návaznosti

- každá etapa je ukončena vnitřní oponenturou
- problém v etapě znamená vracet se k některé z předchozích etap
- každá etapa má své dokumenty (dokumentace se buduje během celého životního cyklu)
- ideálně se nezabývat nižšími etapami při projednávání vyšší
- vystopovatelnost požadavků !!! (CMM)

[vsuvka]

CMM je Capacity Maturity Model. Jde o zdokonalování procesů. Cílem CMM je zvýšení spokojenosti uživatelů SW systémů, zlepšení kvality SW a omezení rizik spojených s vývojem softwaru zpřesněním odhadů při plánování a sledováním průběhu prací. CMM je i nástrojem zlepšování efektivnosti práce firmy a zmenšování rizik. Definuje několik úrovní:

1. **počáteční úroveň**: neformální; jak si kdo co pamatuje; při odchodu pracovníka jsou jeho znalosti ztraceny
2. **opakovatelná úroveň**: jsou zavedená pravidla pro řízení projektů (standardy, zásady);
3. **definované procesy**: standardy od specifikace požadavků až po management procesů a jejich provázanost; součástí norem jsou i nástroje kontroly a zlepšování; školení pracovníků
4. **řízené procesy**: metriky (sběr, vyhodnocování, trendy, chybová místa, statistická analýza); odhad termínů a nákladů
5. **optimalizované procesy**: procesy neustálého vypelšování; tým hodnotící kvalitu procesů; analýza úspěchů a neúspěchů.

Úrovně 4 a 5 (částečně i 3) jsou dosažitelné jen u velkých firem. Nebezpečí byrokratizace vývoje SW.

Produkty jendotlivých etap

Specifikace požadavků

- neformální specifikace
- formální specifikace
  - funkční požadavky
  - nefunkční požadavky
- uživatelský vzhled aplikace
- oponenturou je feasibility study (uskutečnitelnost)

Specifikace systému

- High Level Design dokument
  - analýza funkčních požadavků → co by měl systém poskytnout uživateli za funkce; co se musí udělat (např. uživatel může vyhledat seznam všech objednávek)
    - diagramy použití
    - analytický diagram tříd
    - základní sekvenční diagramy
    - základní komunikační diagramy
  - analýza nefunkčních požadavků → kladou omezení na design a provedení (např. výkon, spolehlivost, bezpečnost, ...)
    - varianty architektury běhového prostředí
    - deployment diagramy
  - hlavní rizika projektu
- předběžný projektový plán
  - základní milestony
  - termíny
  - základní cena
  - time-material vs fixed-priced
- plán testování
- návrh uživatelského manuálu
- oponentura je revize zákazníka + vnitřní oponentura

Návrh

- návrh architektury
  - specifikace architektury
  - plán testování systému
- návrh rozhraní
  - specifikace rozhraní
  - plán integračních testů
- podrobný návrh
  - specifikace návrhu (detailní návrhový class diagram, sequence diagramy, přesný deployment diagram)
  - plán testování jednotek
- běhová dokumentace
  - popis běhového/vývojového/testovacího prostředí

    - plán obnovy po výpadku
    - instalační skripty
    - upgrade skripty
- uživatelská dokumentace
    - uživatelská příručka
    - instalační příručka
    - běhová příručka
    - tutoriály
- oponentura je informace zákazníka + vnitřní oponentura

Impmementace

- balíčky s kódem
- plán testování jednotek
- revize kódu jako oponentura, čtení, inspekce

Testování

- testovací scénaře
- konečný uživatelský manuál
- záznamy o provedených testech a výsledky
    - protokol o testování jednotek
    - protokol o testování modulů
    - protokol integračních testů
    - protokol testování sytému

Předávání systému

- přejímací testování (akceptační testy)
- konečný systém a dokumentace
- předávací protokol

Provoz a údržba

- chybové výstupy aplikace
- sběr uživatelských požadavků a postřehů

3. Aplikace CASE v životním cyklu

Computer Aided Software Engineering (CASE) = počítačem podporované sw inženýrství. Vznikly jako požadavek na automatizaci vývoje. (→ modelovací nástroje)

CASE nástroje primárně umožňují:

- modelování IT systému pomocí diagramů (člověk lépe chápe obrázek než složitě psané slovo),
- generování zdrojového kódu z modelu (usnadňuje práci programátorům),
- zpětné vytvoření modelu podle existujícího zdrojového kódu (reverse engeneering),
- synchronizaci modelu a zdrojového kódu,
- vytvoření dokumentace z modelu.

Např. Rational Rose, Oracle Designer, Case Studio, MS Visio.

Z hlediska životního cyklu SW se nástroje CASE dělí do následujících skupin:

1. **Horní (upper) CASE**
   - použití pro specifikaci cílů, počáteční fáze specifikace požadavků, řízení projektu
   - hlavním cílem je porozumění a specifikace systému jako celku (analýza organizace, v níž se má systém používat, zobrazení procesů v organizaci, definice klíčových informačních toků, atp.)
   - patří sem i nástroje pro plánování a vedení projektů, procesní inženýrství
   - hlavní nástroje
     - diagramy toků dat a jejich varianty
     - ER diagramy bez podrobné specifikace všech atributů
     - prostředky pro řízení projektů
     - dokumentografické systémy
     - popis základních vlastností systému prostředky OO modelování

2. **Střední (middle) CASE**
   - použití pro podrobné specifikace požadavků, návrh systému, dokumentace a vizualizace systému
   - hlavní cíle: formalizace specifikace a návrhu s cílem usnadnění změn a snažší komunikace se zákazníky, vytvoření modelů usnadňujících, případně umožňujících generaci návrhu
   - je jádrem komerčně dodávaných CASE systémů
   - především u větších firem
   - zahrnují prostředky pro podrobnou specifikaci požadavků a návrh systému
   - nejúspěšnější, neboť pokrývá činnosti, které nejsou předmětem činnosti konzultačních firem, a firem zabývajících se tvorbou modelů podniků a řízením projektů
   - nejsou dosud ve větší míre integrovány do moderních prostředí pro vývoj SW
   - hlavní nástroje:

- diagramy toků dat včetně možnosti podrobnějšího popisu procesů, dat. úložišť a datových toků
- ER diagramy s možností detailní specifikace atributů
- pro objektovou metodologii diagramy OO technik – diagramy tříd a jejich vztahů, diagramy instancí, přechodové dia., atp.
- systém správy dokumentů a správy konfigurace
- systémy vyhodnocování metrik souvisejících s návrhem systému a specifikacemi požadavků
- vývoj prototypů, většinou potěmkinovských, návrh rozhraní, generátory obrazovek a sestav
- generátory definic dat. Někdy pouze kostry definic, někdy se doplňují ručně.

3. **Dolní (lower) CASE**
   - obsahují nástroje na podporu kódování, testování a údržby a reverzního inženýrství
   - **hlavní nástroje**
     - generátory kódu
     - prostředky reverse engineering. Nástroje umožňující rekonstrukci dokumentace z existujícího SW nebo alespoň detekci míst, kde již existující dokumentace neodpovídá aktuálnímu stavu.
     - prostředky sledování a vyhodnocování metrik kódu
     - prostředky a nástroje plánování a zajišťování kvality SW (sběr info o průběhu testování, řízení testování, sběr a vyhodnocení dat, inspekcí a výsledků testů, pravidla přijímání prvků konfigurace, podpora plánování opatření na zajišťování kvality)
     - správa konfigurace
     - prostředky sledování a vyhodnocování práce systému

Aktuálnější dělení:

1. Pre-CASE: plánování
2. Upper-CASE: specifikace požadavku
3. Middle-CASE: kooperace při návrhu
4. Lower-CASE: návrh a vývoj
5. Post-CASE: údržba, modifikace

4. Specifikace požadavků

Analýza představuje studium problému před tím, než podnikneme nějaké akce směřující k jeho řešení. Předmět analýzy:

- Existující systém, jehož struktura a funkce nejsou zřejmé zákazníkovi ani řešiteli, případně zákazník neumí strukturu chování systému řešiteli vysvětlit.
- Neexistující systém, o němž má zákazník nepříliš přesnou představu a neumí požadované funkce řešiteli vysvětlit.

Výsledkem analýzy je specifikace systému.

Specifikace požadavků je součástí analýzy. Vychází z dokumentu „Stanovení cílů“ (z fáze stanovení vizí). Obsahuje cíl řešení, požadovaný výsledek. Je základem a úzkým místem každého systému. Cílový stav je dokumentovaný tak, aby bylo možné posoudit, zda implementace uvedeného stavu dosáhla. Dokument Specifikace požadavků je závazným podkladem pro návrh a realizaci systému. Může být formální i neformální.

Specifikace požadavků je ohrožena dvěma faktory: neví se přesně co (potřeby podniku) + zájmy jednotlivých skupin v podniku je potřeba vyvažovat (ale neví se dopředu jak).

Proč si nemůže zákazník specifikovat systém sám: syndrom pejska a kočičky jak vařili dort (ještě tohle a taky tamto …); zřídká zná zákazních možností IT technologií; většinou chybí komplexní znalost fungování podniku; pravděpodobnost že by se prosadily jen zájmy těch co by systém specifikovali.

Je důležité systém specifikovat nejen s vedením, ale také s lidmi, kteří se systémem budou opravdu pracovat (např. skladník). Také je dobré zahrnout do projektu lidi mající zájem o zlepšení své práce, kteří se budou sami aktivně zapojovat a pomáhat nám.

Má obvykle následující strukturu:

1. název projektu a identifikátor projektu
2. úvod - shrnutí úkolů v obecně srozumitelné rovině
3. vymezení uživatelů a způsobu využití produktu
4. perspektivy realizovaného systému (doba života,...)
5. způsob vedení dokumentace
6. zajištění spolupráce mezi dodavatelem a uživatelem
7. dokumenty odkazované v textu
8. použité zkratky a slovník
9. vazby na jiné projekty
10. požadavky na HW, efektivnost a spolehlivost
11. rozpis dat a funkcí - dekompozice systému
12. plán testů
13. vymezení obsahu dokumentace předávané uživateli
14. termíny realizace, plán realizace
15. ekonomické a organizační zajištění (odhad nákladů, řešitelský tým)
16. vymezení způsobu údržby a způsobu prodeje produktu dalším uživatelům

Základní vlastnosti specifikace požadavků:

- **úplnost**: funkce, efektivnost, rozhraní, reakce na chyby, ...
- **ověřitelnost (testovatelnost)**: požadavky musí jít ve finále otestovat
- **bezespornost**: požadavky si nesmí odporovat
- **konzistentnost**: podobné problémy/věci se řeši podobně/stejně
- **modifikovatelnost a srozumitelnost**: lze snadno provést změny
- **vystopovatelnost**: každý požadavek má své důvody
- **použitelnost i během provozu systému**
- **stabilnost**: aby se požadavky neměnily příliš

Techniky zjišťování požadavků

Sběr dat obecně:

- cizí firma: výhoda odstupu, ale drahé
- samotná firma: levné, ale nemusí podchytit všechny procesy (některé intuitivní)
- řešitelská skupina zahrnující oba (doporučuje se)

Jednotlivé techniky:

- **Interview:** dobře připravený pohovor o tom, co pracovník uživatele dělá, co by mohl IS zlepšit či přinést. Může být i skupinové. Existence moderátora a zapisovatele. Při dobrém moderátorovi nejlepší. Je adaptabilní, ale zabere čas a chce to mít k dispozici respondenty.
- **Strukturované interview:** interview, kde se postupně odpovídá na otázky dle předem připraveného dotazníku. Není závislé na moderátorovi (výhoda i neýhoda)
- **Dotazníky:** rozesílají se, uživatelé vyplní sami. Levné, masové, ale nezjistí se vše. Je důležité umět klást správné otázky a taky se ptát na jednu věc chytře vícekrát, abychom si ověřili odpovědi. Ne všichni odpoví a taky ne všichni odpoví správně
- **Studium dokumentů:** používaných zákazníkem.
- **Pozorování chodu prací u zákazníka:** drahé a pracné, zdlouhavé, ruší při práci, ale zjistíme jak to opravdu funguje a ne jak si někdo myslí že to funguje skutečná realita)
- **Účast na pracovním procesu:** totéž co výše, někdy se neodchytí výjimečné případy.
- **Analýza existujícího IS:** nebezpečí převzetí chyb původního IS. Může obsahovat spoustu balastu a zbytečných funkcí. Taky můžeme pak mít snahu použít stávající řešení.
- **Společný vývoj požadavků:** formulace požadavků skupinou pracovníků uživatele a konzultantů dodavatele IS.
    - Brainstorming: Neformální porada s cílem najít nová řešení, vize a myšlenky. Okamžité nápady se hned zapisují na flipcharty a obvykle neformálně hodnotí, i bláznivé nápady se nezatracují a zapisují, vyhodnocení a koordinace nápadů již

nebývá součástí porady.

- Paralelní brainstorming (vlaječky) – rozdělím tým na skupinky → rozumnější šance na uplatnění všech, snazší koordinace
- Úhly pohledu (klobouky) – emoce a intuice, kritika, přínosy, fakta (případně ještě řízení a tvůrčí myšlenky)

- **Workshop:** pro hodnocení a kontrolu průběhu prací, získání přehledu o stavu prací. Členění: úvod, řada kratších vystoupení - presentací výsledků s diskusí, shrnutí a závěr.
- **Oponentury:** nejefektivnější detekce anomálií, snížení rizika neúspěchu. Pracné, detekuji, ale nemám opravovat (revize, inspekce,...).

5. Prototypy a oponentury

Prototypy

SW prototyp je částečně funkční model systému, který realizuje či simuluje některé vlastnosti systému. Prototypování umožňuje odhalit nedostatky již v začátku projektu. Prototyp není určen k cílovému řešení, pouze k ověření specifikace funkcí, zpřesnění požadavků. Bohužel zřídka otestuje vlastnosti, jež se ukážou až v provozu.

Důvody pro vytvoření prototypů:

- ověření správnosti a úplnosti specifikace požadavků
- ověření správnosti a úplnosti funkcí a návrhu struktury systému
- předběžný odhad nákladů a rizik realizace

Výhody: ověření správnosti, lepší konzultace se zákazníkem a odhady pracnosti, brzo máme něco v ruce.

Nevýhody: větší pracnost, ne vždy odhalí funkční chyby.

Typy prototypů

- **Potěmkin** (obrazovkový prototyp): model cílového systému, který simuluje uživatelské rozhraní, tedy obrazovky dialogů a tvar tiskových sestav. Výkonná část systému téměř, čí úplně chybí. Simulace budoucího rozhraní.
- **Neúplný**: modeluje pouze některé funkce.
- **Jiný kůň**: téměř úplný, ale funguje na jiném HW nebo nad jiným základním SW
- **Hlemýžď**: neefektivní, pomalý. Prototyp je realizován v jazyce neumožnujícím cílovou efektivnost (např. Prolog)
- **Nepříjemný, nerudný**: má nepříjemné uživatelské rozhraní nebo je nestabilní
- **Lajdák, záludný**: nereaguje správně na chyby v datech a chyby obsluhy
- **Samotář**: není schopen propojování (Prolog)

Oponentury

- jsou nutné u velkých a kritických aplikací, méně formalizované jsou dobré i u menších projektů
- najdou sice méně problémů než testy, ale zase najdou jiné typy problémů (např. problémy vizí, specifikací, koncepce)
- při správném provedení je nalezení a odstranění defektu pomocí oponentury levnější než pomocí testování

Např. specifikace požadavků by měla být zakončena oponenturou. Kontroluje se splnitelnost požadavků, dodržení záměrů cílů projektu, návrh a zhodnocení plánu dalšího postupu, správnost požadavků, bezespornost, atd. Výstupní dokument je feasibility study (studie splnitelnosti). Její součástí mohou být výstupy částí vnitřních oponentur. Nejpozději se vypracovává před zahájením kódování, nejdříve však po dokončení spec. požadavků.

Typy oponentur:

1. **inspekce**: oponentura menších celků pode přísných pravidel; dobrá inspekce odhalí 80% chyb
2. **revize (review)**: oponentura většího celku; techniky – bežná oponentura, čtení kódu, strukturované procházení
3. **simulace**: procházení a napodobování provádění

Dělení:

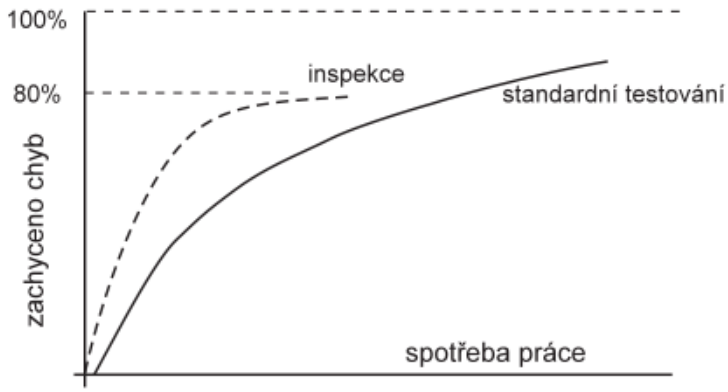
- **vnitřní (dohled):** kontrolní činnosti prováděná vedením firmy formou kontrolních dnů. Prověrka skutečností důležitých z manažerského hlediska.
- **vnější (audit):** varianta porady, která prověřuje dodržování podmínek smlouvy, funkce systému, zda se řešení (ekonomicky) neodchyluje od plánu a zda je naděje na dosažení cílů co do obsahu i termínů. Dohled je prováděný nezávislou organizací. Provádí auditor.

Vnitřní oponentury

- kontrolní akce prováděné členy řešitelského týmu
- jednotlivé ruhy oponentur se liší úrovní formalizace a počtem účastníků
- společné rysy všech oponentur
  - účastní se řešitelé, možno i spoluřešitelé ze strany budoucích uživatel
  - cílem je detekce chyb (přehlednutí chyby je selhání oponentury), chyby se neodstraňují, jen zaznamenávají
  - detekce chyb nesmí být důvodem postihu jejich původců (ptž. postihy snížují účinnost oponentur)
  - neměly by trvat déle než 2 hodiny

(1) Inspekce

- oponentury prováděné podle přesných pravidel v 1 nebo více fázích ve skupině
- **Jednofázové inspekce**
  - tým (3-6 lidí), vedoucí (moderátor), 1-3 oponenti, předčitatel a zapisovatel
  - pročítá se specifikace části nebo dokument nebo program, ostatní hledají chyby
  - materiály mají několik dní předem, o problémech se dělá zápis
  - méně než 2 hodiny (ptž. účastnící pak ztrácejí pozornost), řídí moderátor, nemá se toho účastnit vedení!
  - problémy se neřeší, jen detekují (výjimkou je uživatelská dokumentace)
  - inspekce by měly být prováděny shora dolů (od celku k částem) po jednotlivých úrovních hierarchie dekompozice
  - odhalení až 80 % chyb



Obr. 8.1: Rychlost odstraňování chyb při inspekcích a při testování.

- **Etapy jednofázové inspekce**
  - jmenování moderátora
  - plánování (moderátor), příprava materiálů (splňují kritéria pro inspekci?), výběr členů týmu, termín inspekce
  - úvodní studium (stanovení rolí + co je účelem inspekce)
  - příprava (rozdání materiálů pár dní dopředu a jejich studium materiálů; příprava místnosti, vybavení, ...)
  - vlastní inspekce (pod vedením moderátora, předčitatel čte dokument, zjišťují se chyby, dělá se zápis)
  - vypracuje se zápis (data do DBS)
  - přepracování (pověřené osoby opraví dokumenty)
  - kontrola (kontrola, zda byly všechny objevený chyby opraveny; poté lze vyvolat novou inspekci)

- **Aktivní inspekce**
  - kontrola výše zmíněnými inspekcemi je možná až při vyhodnocování výsledků testů nebo až při předání, což je často pozdě a vždy drahé → metoda aktivní inspekce a „zasetých chyb“
  - sleduje kvalitu inspekcí
  - aktivní inspekce → zadávají se kontrolní otázky (co jak funguje, proč se vybralo to a to, …) → vyšší aktivita a pracovní nasazení inspektorů
  - vhodné pro kontrolu programů a návrh dat
  - zaseté chyby → do oponovaného materiálu se uměle zasejí chyby a pak se zjišťuje, kolik bylo nalezeno zasetých chyb a skutečných chyb

$$\frac{c}{C} = \frac{z}{Z}$$

c … počet nalezených chyb
C … celkový počet dosud neznámých chyb
z … počet nalezených zasetých chyb
Z … celkový počet zasetých chyb

- **Vícefázové inspekce**
  - některé činnosti se při inspekci provádějí separátně, abychom už tak náročnou inspekci trochu zefektivnili a „zlevnili“
  - **Etapy vícefázové inspekce:**
    1. etapa (provádějí jednotlivci):
       - kontroly formálních vlastností (ty lze provést v principu i počítačem) Například se kontroluje celková struktura dokumentu, mnemotechnika zkratek, identifikátorů, index a úplnost odkazů, programovací standardy, typ org. rozložení, ...
    2. etapa (skupinové inspekce):
       - oponenti dostanou materiály předem spolu s kontrolními otázkami jak co funguje
       - oponenti individuálně pročítají program či dokument a řídí se seznamem otázek a pokyny, co mají sledovat
       - ve skupině se výsledky z výše uvedených bodů jednotlivých respondentů porovnávají, zjistí se nejasnosti
       - provede se oponentura
       - vše se zanese do zápisu

(2) Revize

- méně formální než inspekce a lze použít na větší celky
- **Etapy revize:**
  - určí se moderátor, ten si vybere oponenty
  - každý oponent dostane k analýze určitou část oponovaného materiálu a snaží se nalézt problematická místa
  - provede se vlastní revize (stručně se specifikuje úkol, uvedou se a prodiskutují zjištěné nedostatky, zhodnocení dodržení plánu práce, lze vypracovat i doporučení, zhodnocení kvality materiálu)
  - výstupem revize je souhrnné hodnocení s přílohami obsahující seznam problémů
  - Výhody: větší flexibilita, možnost oponovat rozsáhlejší materiály, menší nároky na kvalitu členů oponentského týmu.
  - Nevýhody: menší účinnost, menší možnosti měření kvality provedení.

Další techniky oponentur:

- **Ve dvojicích**: dvojice (extrémní programování), vedoucí týmu vs. jeho zástupce. Jeden vymýšlí a píše a druhý kontroluje. Je to efektivní, zlepšují se znalosti, možnost převzetí.
- **Procházení nebo strukturované procházení (walkthrough)**: dvojice až trojice,

jeden vysvětluje ostatním, často sám je schopen pak chybu odhalit. Podmínkou je dobrý vztah mezi členy, vysoké pracovní nasazení.

- **Simulace textů (čtení kódu)**: u programů se simuluje chování programů; prevence, obtíže detekce defektů, dnes již částečně řeší ladící programy, ale neřeší vše.
- **Cleanroom**: formalizovaná metoda, zahrnuje formální důkazy správnosti, při vývoji IS ne příliš efektivní, lépe tam, kde netřeba úzce jednat se zadavateli.
- **Týdenní posezení u kafe**: pravidelné schůzky, neformální diskuse, podpora dobrých vztahů, odchycení vznikajících problémů.
- **Oponentury zdrojových textů programů:**
  - shora/zdola/efektivně
  - systém je hierarchie daná vztahem A potřebuje B: A → B
  - pokud je dobře navržen, bývá to hierarchický strom
  - lze se rozhodnout, že začneme od A po směru šipek (shora od kořene), vč. potomky (do šířky) nebo
  - zdola od listů (raději do šířky) nebo
  - selektivně shora/zdola se snažíme co nejdříve dostat k oponentuře problematických komponent
  - **Ne/výhody:**
    - **shora**: oponují/testují v prostředí, které se bude skutečně používat, ale menší obecnost
    - **zdola**: větší pracnost, obecnost, náročnost na data a předpoklady.
- U oponentur lépe postupovat shora, u testů to není tak jednoznačné.

Činnosti pro zajištění kvality IS:

- **Evaluace**: celkové zhodnocení (materiálů, alternativ, …). Provádí se technikou revize nebo inspekce. Hlavní cílem je oveření, zda jsou požadavky úplné a shodné s cíli projektu
- **Verifikace**: jsou specifikace v souladu s cíli + je návrh s v souladu se specifikací + je kód v souladu s návrhem?
- **Validace**: funguje to správně? Ověřuje se testováním
- **Audit**: nezávislé proveření dodržování dohod a stavu plnění úkolů

Vlastnosti členů oponentských týmů:

- **Moderátor**: nestranný (ne autor!), musí motivovat ostatní
- **Předčítající**: musí co nejpřesvědčivěji prezentovat materiál
- **Zapisovatel**: puntičkář, musí zachytit vše podstatné
- **Oponent**: objektivní, nesmí na nikoho útočit, měl by mít snahu přispět k řešení

Závěr

- Kniha IS (Král):
  - Životní cyklus SW + vodopád str. 26
  - Systémy CASE str. 297
  - Oponentury str. 103
  - Varianty procesů vývoje software str. 97
  - Zjišťování požadavků str. 87
- Otázky TIS I a TIS II
- ANANAS slidy z přednášek