Základní modely životního cyklu software, softwarový proces, metodika

Životní cyklus (ŽC, LC)

Proces od zahájení vývoje SW až po jeho vyřazení z provozu.

ŽC SW je model procesu provozování SW systému, definuje postup ve fázích:

- Phasing-in – postupné zavádění produktu (do doby nasazení)
- Phasing-out – postupné vyřazování produktu (od doby nasazení)

Fáze ŽC vývoje SW:

- Analýza požadavků
- Návrh systému
- Implementace a testování
- Integrace a nasazení
- Provoz a údržba

Softwarový proces

Systematická série akcí vedoucí k (vyšší pravděpodobnosti dosažení) výsledku: kvalitnímu softwaru.

- **Členění: fáze, aktivity**
  - Aktivity: komunikace, plánování, modelování, konstrukce, nasazení; řízení, kontrola kvality, dokumentace
- **Mezivýsledky: artefakty**
  - Technické: specifikace, kód, dokumentace, testy
  - Obchodní: plán, rozpočet, prodatelný produkt
  - Komunikační: plán, specifikace
- **Činitelé: role**
  - Technické (analytik, vývojář, tester)
  - Manažerské (ředitel, project manager, team leader)
  - Podpůrné (poradce, kouč, sekretářka…)

Varianty:

- **Sekvenční:** vodopád
- **Cyklické:** spirála (opakování aktivit, produkt roste)
- **Iterativní:** RUP
- **Agilní:** SCRUM, XP

Metodika

Metodika = definovaný proces pro konkrétní účel, tj. fáze, aktivity, role, artefakty, milníky atd. jsou dobře popsány

- Booch method
- SSADM, RUP, SCRUM
- UML není metodika!

Sekvenční a iterativní vývoj SW, výhody nevýhody, důsledky, způsoby dodávky produktu.

Sekvenční

Vodopád

- **Přístup k vývoji SW jako k přípravě výrobku na pás – každý krok jednou, až když je úplně „dokonalý“, pokračuje se dalším krokem**
- Metoda je striktně formulována a vede návrháře při jeho práci pomocí jednoznačně definovaných kroků a dílčích cílů
- Systematicky se prověřují dosažené cíle a provádí se jejich korekce
- Vychází z datového modelu; základní datový model se snaží zformulovat už v počátečních etapách
- Odděluje logický a fyzický návrh systému
- Již v průběhu analýzy by měli být podchyceny nestandardní situace a chybové stavy systému
- Vhodný pro projekty s předem známou problematikou a neměnnými požadavky
- **Výhody**
  - snadné k pochopení
  - dobrá možnost řízení a sledování postupu řešení (milníky(milestones))
  - klade důraz na dokumentaci -- specifikace, design, analýza
- **Nevýhody**
  - Velké riziko selhání
  - vyžaduje mít na počátku přesně a úplně definované požadavky (uživatel často nedokáže stanovit předem)
  - provozuschopnost verze vidí zákazník až v závěrečných fázích řešení, případné závažné nedostatky jsou odhaleny velmi pozdě.
  - během vývoje se mohou měnit požadavky a výsledkem je, že dodaný produkt není to, co zákazník chtěl
  - během implementace se zjistí, že design není v pořádku a je třeba ho změnit

Iterativní

Přírůstkový vývoj: Rozbití velkého projektu na několik malých projektů (protože pro malé projekty vodopád funguje dobře) které na sebe postupně navazují a přidávají funkčnost, dokud nejsou splněny požadavky na produkt. Zavedení iterace jako samostatného projektového celku (podle sekvenční metodiky) s vlastními cíli, postupy, testy a hodnocením.

Popis: několik vodopádů za sebou.

Způsob dodávky produktu

Velký třesk

- malé projekty, jasné požadavky

Přírůstkově

- určení přírůstků -> plán -> postupné dodávky
- zpětná vazba, ale úpravy projektu obtížné

Evolučně

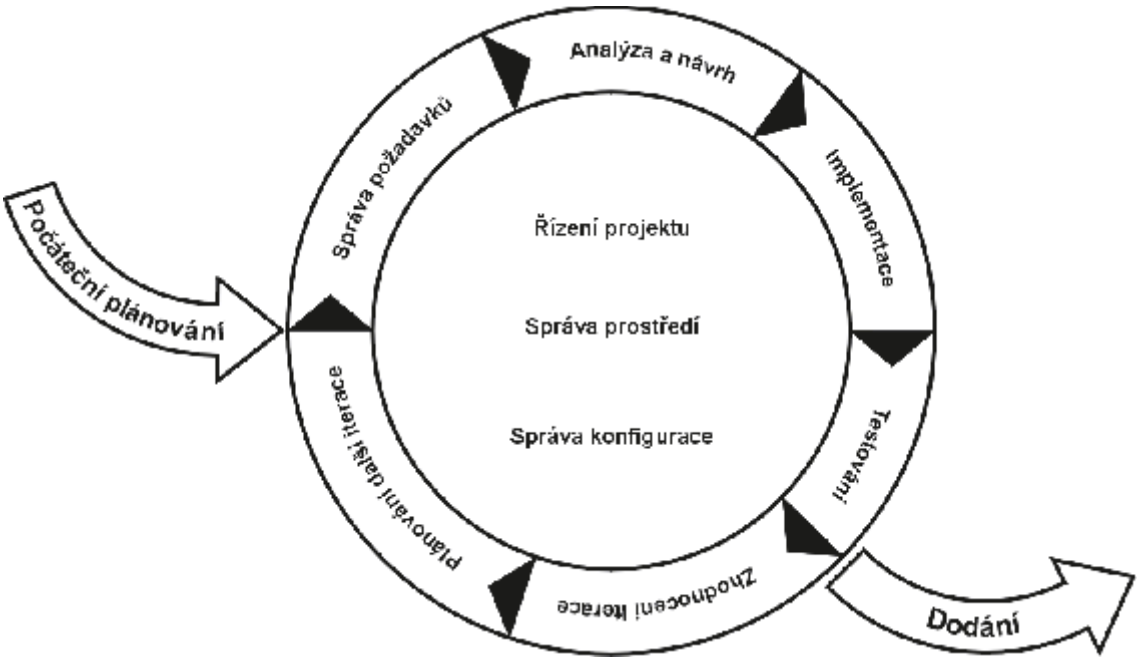
- cyklus: určení cíle -> dodávka -> zpřesnění ("growing sw") - agilní metodiky

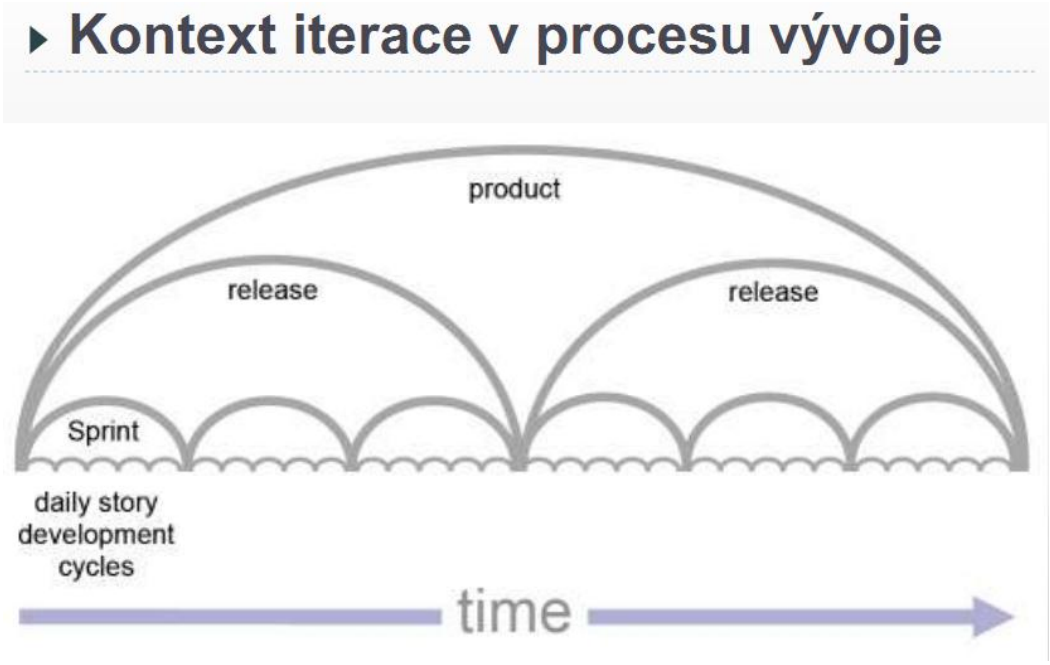
Krabicový software: příjemce produktu (a oponent projektu) je uvnitř firmy – produktový manager, obchodní oddělení. Software se zákazníkovi dodá hotový v krabici, ale uvnitř podniku lze volit způsob

„předání k prodeji“. => Software do krabice se nemusí dělat velkým třeskem, i když to definičně odpovídá.

Vlastnosti iterace, její průběh

Iterace = malý vodopád.





Vlastnosti

- **Omezení v čase:** datum konce iterace je vždy pevně stanoveno (**timebox !**)
- **Délka:** malá je lepší – blízký cíl, menší riziko, rychlost adaptace, blíž je plánování další iterace (změny)
- **Neměnnost:** probíhající iterace je uzavřená změnám zvenčí, ty lze zohlednit až v plánování další iterace

- **Cíle a milníky:** jednoznačně definované cíle každé iterace, meziprodukt
- **Charakter podle fáze celého projektu:** Zahájení, projektování, konstrukce, nasazení

Průběh

- Plánování
- Sestavení požadavků / výběr z požadavků
- Analýza a návrh
- Implementace
- Testování
- Zhodnocení iterace, podklady pro další iteraci

Příklad sekvenční, iterativní a agilní metodiky

Sekvenční : vodopád

Iterativní: Spirála

Agilní: SCRUM, XP

SCRUM vs. XP

Jsou si hodně podobné, SCRUM vychází z XP
SCRUM: delší sprint, XP: kratší iterace
SCRUM: vyšší úloha managementu (SCRUM MASTER), XP: párové programování, test-driven, neformální šéf (role může putovat po týmu)
SCRUM: úlohy si programátoři berou, nezáleží na pořadí; XP: pevné pořadí úloh
Často se používají dohromady nebo se kombinují

Pozor: SCRUM Master není team leader, je to jenom „ceremoniář“

Požadavky na software – typy požadavků, formy popisu, úrovně detailu a jejich vztah k procesu vývoje

Co je to požadavek?

- *požadavek* = schopnost nebo vlastnost, kterou má sw mít, aby jej uživatel mohl použít k vyřešení problému nebo dosažení cíle, který vedl k zadání, nebo aby splnil podmínky stanovené smlouvou, standardem nebo jinou specifikací.
- vlastnosti požadavku: úplný, bezesporný
- požadavkem není to, co uživatel nepotřebuje

Typy požadavků

- **Funkční požadavky** = funkce
  - popisují funkce nebo služby, které jsou od systému očekávány
  - př.: požadavky na univerzitní knihovní systém
    - systém by měl poskytovat uživatelům vhodné rozhraní pro čtení dokumentů v úložišti dokumentů
- **Mimofunkční požadavky** = vlastnosti
  - netýkají se funkcí systému, ale vlastností jako je spolehlilvost, čas odpovědi, obsazené místo na disku nebo v paměti, aj.

        - často kritičtější než jednotlivé funkční požadavky (např. pokud je řídící systém letadla nespolehlivý, je nepoužitelný)
        - někdy dané vnějšími faktory, tj. legislativní požadavky (př. zákon na ochranu osobních údajů apod.)
        - př. veškerá komunikace mezi uživatelem a systémem by měla být vyjádřitelná ve znakové sadě ISO 8859-2
- **Business požadavky**
    - Vize a rozsah projektu
    - Smluvní záležitosti
    - Náklady, **TCO**
- **Právní a další**
    - Např. různé právní systémy dvou zemí a cloud

Způsob formulace požadavku:
- **uživatelská specifikace**
    - vysokoúrovňový popis funkčních a mimofunkčních požadavků zákazníka
    - musí být srozumitelné pro uživatele, kteří nemají technické znalosti
- **systémová specifikace**
    - podrobnější specifikace uživatelských požadavků pro vývojáře
    - slouží jako výchozí bod pro design systému

Formy popisu
- textový popis
    - shopping list
    - strukturovaný text
- grafické vyjádření
    - use case diagramy
    - ERA, UML
- implementace
    - popis ve formě prototypu a uživatelské příručky

Úrovně detailu v rámci procesu vývoje

- **Zahájení projektu**: strategické, klíčové, obrysy
- **Projektování**: podstatné, úplnost
- **Konstrukce**: podrobnosti

Úroveň detailu a agilní metodiky

Dokument specifikace požadavků (DSP)

- konečný výsledek **analýzy požadavků**
- kompletní **popis chování systému**
- zahrnuje případy užití popisující všechny interakce uživatele se SW -- funkční požadavky
- technický dokument, oficiální vyjádření o tom, co se od vyvíjeného systému očekává (dohoda mezi zákazníkem a dodavatelem, co má zadaný sw dělat a jak to má vypadat)
- základ pro pozdější **ověření správnosti** - důraz na jednoznačnost, **ověřitelnost**, reálnost, srozumitelnost, úplnost, přesnost a správnost, modifikovatelnost, konzistenci
- měl by specifikovat pouze externí chování systému, tj. snaha **vyloučit design** z DSP
- strukturován tak, aby v něm bylo snadné provádět změny (modifikovatelnost)
- měl by specifikovat omezení implementace -- mimofunkční požadavky
- měl by charakterizovat přijatelné odpovědi na nežádoucí události

Jednodušeji:

- jednoznačnost
- úplnost
- srozumitelnost
- modifikovatelnost
- přesnost
- ověřitelnost
- reálnost
- specifikace pouze chování - NE jak to udělat

Postupy pro sběr požadavků

Problematická a ošemetná to záležitost:

- Uživatelé nerozumí tomu, co chtějí, nebo uživatelé nemají jasnou představu o svých požadavcích
- Uživatelé neschválí seznam sepsaných požadavku jako finální
- Uživatelé trvají na nových požadavcích i po zafixovaní nákladů a časového harmonogramu
- Komunikace s uživateli je pomalá
- Uživatelé se často nepodílejí na kontrolách, nebo jsou neschopní to udělat
- Uživatelé jsou technicky nevzdělaní
- Uživatelé nerozumí procesu vývoje
- Uživatelé neví o současné technologii

⇨ Je potřeba předem počítat s různou úrovní počítačové gramotnosti. **To může vést k situaci, kdy uživatelé průběžně mění své požadavky, i když systém nebo vývoj produktu byl zahájen.**

Způsoby sběru požadavků

- *interview*
  - předem připravený rozhovor, který vede moderátor (klade otázky, dává slovo)
  - nedoporučuje se více než 2 hodiny
  - předem si připravit scénář, které okruhy se budou probírat, v jakém pořadí, scénář se snažit nenásilně dodržovat
- *pozorování, práce s uživateli*
  - pozorování prací u zákazníka (účast analytiků)
- *analýza existujícího systému*
  - inspirujme se tím, jak funguje stávající systém
- *dotazníky*
  - vhodnými otázkami zjistíme od uživatelů, co potřebují
- *prototypování* – tvorba prototypů, podle kterých si zákazník ujasní své požadavky
  - stačí na papír nebo skutečné programové prototypy
- *studium hlášení problémů*

Způsoby vyjádření

- *přirozený jazyk*
  - výhodou je srozumitelnost pro uživatele

        - nevýhodou – spoléhá se na to, že autoři používají stejná slova pro stejný koncept (stejná věc se dá říci mnoha různými způsoby). Obtížná modularizace - kterých všech dalších požadavků se změna dotkne.
- *formuláře*
    - pro vyjádření požadavku se nadefinuje jeden nebo více typů formulářů
    - měl by obsahovat:
        - popis specifikované funkce nebo entity
        - popis vstupů, odkud se berou
        - popis výstupů, kam putují
        - jaké další entity specifikovaná funkce nebo entita používá
        - případné pre/post conditions (co platí při vstupu do funkce a co při výstupu z ní)
        - pokud vznikají postranní efekty, pak jejich popis
- *pseudokódy*
    - v přirozeném jazyce těžko vyjádřitelné vnořené podmínky nebo smyčky
    - jazyk s abstraktními konstrukcemi, které právě potřebujeme
    - vnoření konstrukcí je vyjádřeno odsazením
    - vyhýbáme se syntaktickým konstrukcím cílového programovacího jazyka (popisujeme požadovaný záměr, nikoli jak to bude v cílovém jazyce)
    - na druhou stranu musí umožňovat téměř automatickou konverzi do kódu
- Obrázky, protoyp GUI

Kontrola požadavků

- musíme zjistit, zda jsou požadavky úplné, konzistentní a zda odpovídají tomu, co zadavatel chce
- vstupem je úplný **Dokument specifikace požadavků**
- metody:
    - přezkoumání (reviews) – požadavky jsou systematicky kontrolovány týmem, manuální proces
    - prototypování – zákazníkovi předvedeme spustitelný model systému
    - generování testovacích případů – vytvoříme testy požadavků, pokud je obtížné vytvořit test, bude požadavek obtížně implementovatelný
    - automatická analýza konzistence – pokud byly požadavky specifikovány jako model ve formální nebo strukturované notaci

Management požadavků

- požadavky na systém se stále mění
- měl by začít plánováním, ve kterém se rozhodne:
    - **způsob identifikace požadavků** – každý požadavek by měl mít jednoznačné ID
    - **proces změny požadavků** – definujeme proces, abychom se ke změnám požadavků chovali konzistentním způsobem
    - **sledovatelnost**
        - zdroj požadavku – kdo požadavek navrhnul, důvod; abychom se mohli zdroje zeptat na podrobnosti
        - vztahy mezi požadavky – kolika požadavků se změna dotkne
    - **nástroje** – co se použije pro uchování informací o požadavcích (malé projekty – obvyklé prostředky(textové nástroje, EXCEL, databáze, aj), velké projekty – CASE nástroje)

Analýza požadavků a tvorba objektového návrhu - postup, použité modely a diagramy (strukturální, UML).

Analýza požadavků obsahuje tři typy aktivit:

- **Sběr požadavků**: komunikace se zákazníky a uživateli za účelem získání jejich požadavků na systém.
- **Analýza požadavků**: identifikování nejasných požadavků, nekompletních, nejasných, nebo protichůdných a následně řešení těchto nesrovnalostí.
- **Zaznamenání požadavků**: dokumentování požadavků v různých formách, jako běžný textový dokument, případy užití (use case), nebo specifikace procesů – > **dokument specifikace požadavků**

Analýza a klasifikace požadavků

- Identifikace zúčastněných stran ("Stakeholderů")
- Případy užití (Use Case)
- XP – user stories

- Funkční
- Mimofunkční
    - Výkonnostní
    - Designové
    - Zákonný rámec

Metoda FURPS – model klasifikace funkčních a mimofunkčních požadavků

- F (functionality) – funkčnost
- U (usability) – užitečnost
- R (reliability) – spolehlivost
- P (performace) – výkon
- S (supportability) – rozšiřitelnost

Podle úhlu pohledu

- **Stakeholder**
    - **Sponsor**
    - **Uživatel**
    - **Oponent**
- **Vývojář**

Další

Agilní metodiky

Agilní metodiky zpochybňují potřebu rigorózního popisu, kategorizace a charakterizace požadavků. Vývoj softwaru považují za pohyblivý cíl. Uživatelské příběhy (user stories) a akceptační testy

Diagram aktivit

Diagram aktivit (UML)

- **akce** – atomické dále nedělitelné kroky
- **vnořené aktivity** – volání jiných procesů (aktivit), tyto aktivity mohou být reprezentovány dalším **diagramem aktivit**.

Sekvenci jednotlivých kroků v diagramu aktivit určuje řídicí tok.

Strukturální model

- Postup se zaměřuje na data a jejich transformaci pomocí procesů systému,
- hlavními nástroji jsou tedy DFD popisující procesy a toky dat a ERD (Entity Relationship Diagram) popisující data a vztahy mezi nimi
- Zaměřuje se na vytvoření logického modelu nového navrhovaného systému (**esenciální model**) a následně přizpůsobení implementačním požadavkům (**implementační model**).
- Model **prostředí**
  - Definuje hranice systému a okolí
  - Obsahuje kontextový diagram a seznam událostí (event list)
- Model **chování**
  - Definuje vnitřní chování systému, tak aby plnil požadavky okolí
  - Používané modely (diagramy) DFD, ERD, DD (datový slovník), SP (specifikace procesů), případně STD (stavový diagram)

Objektový model

- Často se používá **modelovací jazyk UML**
  - Model případů užití
  - Doménový model
  - Diagram tříd
  - Stavové diagramy, sekvenční diagramy a další
- **CRC karty** (Class-Responsibility-Collaboration cards)
  - Umožňuje zvládnout návrh i složitých a velkých systémů
  - Není na první pohled vidět kudy vedou vztahy, musí se vyčíst z karet (barevné rozlišení, čáry na nástěnce…)
- Hierarchie objektů – sdružovány podle logických souvislostí do balíků a podbalíků
- Přirozený přechod od analýzy k návrhu

Doménová analýza

- Doménová analýza nepřihlíží podstatně k jednomu účelu a způsobu použití (kontextu použití), nýbrž shromažďuje pojmy a vazby pro doménu podstatné.
- Hledají se objekty, operace a vazby, které znalci z problémové oblasti pokládají za důležité (často používají jejich názvy apod.)

Architektura softwarových systémů, význam a součásti architektury, architektonické styly.

Význam

- Architektura definuje konceptuální integritu systému.
- Systém má vždy právě jednu architekturu (může integrovat více stylů)
- Definice architektury je první krok návrhu
- Umožňuje myšlenkové pochvopení návrhu velmi složitých systémů
- Stanovuje základní kameny návrhu a základní směry vývoje a údržby

Součásti

- konvence a politiky (pravidla pro návrh, dodržují všichni vývojáři)
- **Funkční, procesní, datová, aplikační**
- členění, doménová analýza:
  - **logické členění** (např. do balíků)
    - balík – skupina souvisejících tříd, tvořící organizační celek, mapování do jazyka (balík vytváří jmenný prostor), hierarchické vnořování
    - třídy v balíku funkčně příbuzné
    - vhodné protože bude přehled o systému a snadné rozdělení implementace mezi členy týmu
    - analytický model tříd je příliš rozsáhlý -> lepší jej členit
  - **funkční členění do subsystémů**
    - subsystém = skupina souvisejících balíků a/nebo tříd tvořící funkční celek
    - vhodné, protože monolitická aplikace není praktická
    - jak najít subsystémy?
      - buď je to dopředu zřejmé (jednoduché, architektonické styly)
      - na základě objektového modelu (nutno vidět všechny třídy a vazby, pak shluk těsně vázaných tříd je kandidátem)
      - na základě případů užití

Architektonické styly

- **Vrstvení** - funkce jsou uspořádány do několika vrstev tak, že funkce vyšší vrstvy mohou využívat pouze funkcí podřízených vrstev.
  - **Monolitická**
  - **Dvouvrstvá (lehký/těžký klient)**
  - **Třívrstvá** je nejběžnějším případem vícevrstvé architektury.
    - **Prezentační vrstva**
    - **Aplikační vrstva (též Business Logic)**
    - **Datová vrstva**

  Porovnání třívrstvé s architekturou MVC
  Model-view-controller má trojúhelníkovou topologii (ne třívrstvou) – pohled je obnovován (aktualizován) přímo modelem, na příkaz řadiče.

Architektura distribuovaných systémů
- Klient-server
- Peer to peer

Filosofický přístup k architektuře/ jednotící architektura: **Service-oriented architecture (SOA)**

Konfigurační management, jeho součásti a role ve vývoji software, základní postupy.

- **Konfigurační management**: „Proces identifikování a definování prvků systému, řízení změn těchto prvků během životního cyklu, zaznamenávání a oznamování stavu prvků a změn, a ověřování úplnosti a správnosti prvků.“ (IEEE)

  ⇨ **Administrační a manažerský aspekt Softwarového procesu**

- **Prvek konfigurace** – Configurable Item (CI)

- konstituující složka systému (konfigurace se sestává z prvků konfigurace)
- jsou atomické z hlediska změn a označování verzí, jednoznačně identifikovatelné
- př.: dokument, zdrojový soubor, knihovna, skript, testovací data, …

- **Konfigurace**
  - SW konfigurace – souhrn prvků konfigurace reprezentující určitou podobu daného SW systému
  - V konfiguraci musí být vše, co je potřebné k jednoznačnému opakovatelnému vytvoření příslušné verze produktu (včetně překladačů, build scriptů, inicializačních dat, dokumentace)
  - Konzistentní konfigurace – konfigurace, jejíž prvky jsou navzájem bezrozporné (tj. zdrojové soubory lze přeložit, knihovny přilinkovat, …)

Role ve vývoji SW

- **Určení a správa konfigurace**
  - určení (identifikace) prvků systému, přiřazení zodpovědnosti za správu
  - identifikace jednotlivých verzí prvků
  - kontrolované uvolňování (release) produktu
  - řízení změn produktu během jeho vývoje
- **Zjišťování stavu systému**
  - udržení informovanosti o změnách a stavu prvků
  - zaznamenávání stavu prvků konfigurace a požadavků na změny
  - poskytování informací o těchto stavech
  - statistiky a analýzy (např. dopad změny, vývoj oprav chyb)
- **Správa sestavení (build) a koordinace prací**
  - určování postupů a nástrojů pro tvorbu spustitelné verze produktu
  - ověřování úplnosti, konzistence a správnosti produktu
  - koordinace spolupráce vývojářů při zpracování, zveřejňování a sestavení změn

Základní postupy

- **Identifikace konfigurace:** stanovení výchozího bodu (baseline, třeba každá major verze), známá kvalita (např. seznam bugů dané konfigurace, kompletní testování před stanovením výchozího bodu), kompletně opakovatelná
- **Řízení konfigurace:** rozhodování o způsobu změny konfigurace a koordinace změny konfigurace po schválení změny změnovým managementem (zm. mgmt schválí, chg. mgmt zavádí)
- **Sledování stavu konfigurace:** dokumentování stavu konfigurace každého vydání a změn konfigurace mezi vydáními (průběh změn mezi jednotlivými výchozími body)
- **Auditování konfigurace:** ověření, že nový výchozí bod implementuje všechny plánované a schválené změny, že nová verze je kompletní, a že dodávka je kompletní vč. právních opatření, dokumentace a dat.

! Role konfigurace ve vývoji vychází z postupů nebo naopak

ITIL: Konfigurační management IT infrastruktury podniku

- zaznamenávání informačních aktiv podniku, nastavení organizace a jejích služeb
- poskytování přesných informací o nastavení procesů a jejich dokumentace,
- poskytovat základ pro Incident Management, Problem Management, Change management a Release Management
- ověření konfiguračních záznamů oproti skutečnosti a jejich sladění.

Standardy

Standardů pro Configuration management existuje poměrně velké množství. Drtivá většina standardů je založena na metodice ITIL. Příklady některých standardů jsou: **IEEE**, ISO, ANSI, NATO standards.

Správa verzí, možnosti verzování, typické situace při správě verzí (větvení, značkování), nástroje pro správu verzí, vazba na správu změn.

Správa verzí

- správa verzí je součást úlohy konfiguračního managementu – identifikace konfigurace
- účelem je udržení přehledu o podobách prvků konfigurace
  - verze popisuje jednu konkrétní podobu
  - v úložišti jsou skladovány všechny verze
- druhy verzí
  - evoluční = revize (př. Word 6.0)
  - alternativní podoba = varianta (př. Word pro Macintosh)
- určení konkrétní verze
  - verzování podle stavu (verze prvku) – identifikují se pouze prvky
  - verzování podle změn (identifikace změny prvku) – identifikují se také změny prvků, výsledná verze prvku vznikne aplikací změn
- granularita
  - celá konfigurace
  - jednotlivé prvky (CI – configurable item)
- popis verze
  - *extenzionální verzování*: každá verze má jednoznačné ID
    - major.minor + build schéma- např. 6.0.2800.1106 (MSIE 6)
    - kódové jméno: One Tree Hill (= Firefox 0.9)
    - marketingový: Windows 95
  - *intenzionální verzování*: verze je popsána souborem atributů
    - např. OS=DOS and UmiPostscript = YES
    - C preprocesor umožňuje intenzionální stavové verzování - např. chceme variantu foo.c pro případ OS=DOS and UmiPostscript=YES

Možnosti verzování

- Rychlý přístup k jakékoli historické nebo alternativní verzi
- Možnost vytvoření branch, tagu => částečná izolace ale s možností aplikace vývoje v trunku
- Pojmenovávání milestounů
- Celý tým má přístup k aktuálnímu stavu vývoje
- Aktuální stav vývoje je jednoznačně určen
- Soukromý pracovní prostor v rámci nejnovější nebo vybrané verze
- Možnost testování lokální změny a commitu až funkční a otestované součásti
- Možnosti pro řešení konfliktů
- Některé verzovací systémy jsou inherentně zálohovací (GIT)

Typické situace při správě verzí (větvení, značkování)

- **Trunk:** Hlavní vývojová větev

- **Branch:** Větve – „soukromý“ vývojový prostor
- **Merge**: Sloučení větve a do kmene a řešení konfliktů
- **Tag:** Značkování – označování milestounů, release apod.
- **Kolize a konflikty** – diff, sloučení dvou konfliktních commitů…Nástroje pro správu verzí

- **Centralizované**
    - **RCS:** revision control systém
        - **Pesimista** je to
        - Pracuje s **jednotlivými soubory**, nepodporuje projekty
        - Historie všech změn vč. autorů
        - Ukládá rozdíly
        - Umožňuje zamykání
    - **CVS:** current versioning systém
        - Práce s **celými konfiguracemi a projekty** najednou
        - **Optimista** – slučování změn
    - **SVN:** subversion
        - Velmi podobný CVS (následník)
        - **Verzuje celé úložiště** (imkrementální číslování revizí)
        - Souborová struktura
- **Decentralizované**
    - Každý uživatel má kompletní lokální kopii repozitáře (klony)
    - Lokální commity, na centrální server lze nahrát víc commitů najednou
    - **GIT** – jádro linuxu
        - Velmi nelineární vývoj, recenzování a začleňování
        - Nelze měnit historické verze
    - **Mercurial –** Netbeans, OpenJDK, Symbian OS
    - **Bazaar –** Ubuntu

Vazba na správu změn

- Vazba revize na ticket/change request
- Možnost požadavků na opravu / update konkrétních verzí (např. long-term support)

Typy požadavků na změny, postup jejich zpracování, nástroje pro podporu řízení změn, vazba na správu verzí.

- **Požadavek na novou funkci/vlastnost**
- **Bug**

Postup zpracování změny

- vytvoření/přijetí (přiděli se ID)
- vyhodnocení (možná řešení, jejich dopady a odhad pracnosti)
- rozhodnutí
    - způsob vyřízení (vyřešit/odmítnout/duplikát/odložit)
    - závažnost (kritická chyba/problém/vada na kráse/vylepšení)
    - priorita (vyřídit okamžitě/urgentní/vysoká/střední/nízka)
- přidělení odpovědné osobě / teamu
- zpracování

- uzavření
    - build: ověření konzistence; verzování: vytvoření nové baseline
    - Informovat zadavatele hlášení a další zájemce

Nástroje

- Bug tracking (BT) systémy
- evidence, archivace požadavků (Ticket systém)
- sledování stavu požadavku (BT, Ticket systém)
- přehled, reporty, grafy, statistiky
- realizace: emailové, webové, klientské
- př. Mantis, Bugzilla, Flyspray, Trac, JIRA

Change Control Board (CCB)

- skupina členů projektu, která má zodpovědnost za změnové řízení
    - vyhodnocování a schvalování hlášení problémů
    - rozhodování o požadavcích na změny (může významně ovlivňovat podobu a chod projektu)
    - sledování hlášení a požadavků při jejich zpracování
    - koordinace s vedením projektu
- složení
    - jedinec – vývojář, QA osoba
    - tým – technické i manažerské role (vhodné, pokud má změna mít velký dopad)

Vazba na správu verzí

- Vazba ticketu/change requestu na verzi
- Vytvoření nové verze s opravou

Sestavení produktu, postup sestavení a jeho varianty, nástroje pro sestavení.

Aktivity podporující transformaci zdrojových prvků do samostatných artefaktů (spustitelných programů/systémů). **Nejdůležitějším krokem je kompilace.**

- Cílem je vytvořit systematický a automatizovaný postup sestavení
- *Vlastnosti sestavení*
    - jedinečnost a identifikovatelnost (buildovací číslo)
    - úplnost - kompletní systém, obsahuje všechny komponenty
    - konzistence - správné verze komponent
    - opakovatelnost - možnost opakovat sestavení kdykoliv v budoucnu
    - dodržuje pravidla vývojové linie
- Typy sestavení
    - *Podle použitých částí*
        - čistý
        - úplný
        - inkeremntální
    - *Podle účelu*
        - *Soukromé sestavení*
        - *Integrační sestavení*
        - *Release build*

Nástroje

- *Make* -- skript popisující závislosti, pravidla, cíle a příkazy. Soubor makefile, program make.
  - Rozšířen hlavně na unixových OS
  - existují různé multiplatformní varianty a platformně specifické příkazy
- *Ant* -- rozšířený nástroj podobný *make* určený pro Javu. Elementy project, target, task. Soubor build.xml
- ***Maven***
  - pokročilý nástroj pro sestavování a řízení projektu
  - především Java platforma
  - oproti Antu zaměřen na konvenci (zdrojáky na předem určeném místě apod.) namísto konfigurace
  - ke kompilaci využívá *Ant*, ale navíc zvládá stahování potřebných knihoven k sestavení aplikace

Způsoby prevence chyb v software, metriky a oponentury

Způsoby prevence chyb v SW

- automatické testování, jednotkové testování
- prověření meziproduktu nezávislým oponentem dříve než se z něj začne vycházet v další práci
- technická oponentura
- párové programování (XP),
- refactoring (bezpečnější než hromadné přejmenování)
- peer review (kontrola nezaujatým čtenářem)
- strukturované procházení (lehčí, flexibilnější verze technické oponentury)

Metriky

- Kvantitativní ukazatele = měřitelná charakteristika nějaké entity.
  - Pomáhají najít slabiny --> zlepšení
  - Dávají přehled a kontrolu nad projektem–produktem
  - Kalibrují odhady
- Výhody
  - Přesnost a dokazatelnost
  - Možnost statistik a vizuální prezentaceZískána na základě dat.
- Metriky samy o sobě k ničemu => musí se provádět měření
- Plán měření – pro projekt
  - **Co, proč, kdy a jak** měřit
  - Jak naměřená data vyhodnotit

Typy metrik produktu

- Složitost, přehlednost
  - Počet možných cest skrz zdrojový kód
  - Fan-in / fan-out => stabilita
    - strukturální metrika, která měří poměr počtu modulů, které volají daný modul ku počtu modulů, které volá daný modul
  - Weighted Methods per Class
    - součet složitosti všech metod ve třídě
  - Lack of cohesion
    - nedostatek soudržnosti - jedna metoda dělá více funkcí (které se ve svém "smyslu" liší)
- **Velikost**

        - **Počet Use Cases**, funkčních bodů
        - **Lines of Code**
            - SLOC (Source Lines of Code),
            - DSLOC (Delivered Source Lines of Code),
- **Kvalita (nepřímé metriky)**
    - Pokrytí testy – kódu, požadavků
    - Charakteristika defektů – hustota, výskyt
    - Kvalita zdrojového kódu
- **Spolehlivost**
    - Střední doba mezi poruchami
    - dostupnost

Způsoby detekce chyb v software, metody testování, vztah k sestavení produktu.

Způsoby detekce chyb

- Chyby ve zdrojáku – odhaleny při překladu
- Statické / Dynamické

- Ladění
- Testování
- Inspekce kódu
- Formální verifikace – automatické ověření zda systém splňuje požadavek

Metody testování

- Whitebox
    - máme k dispozici zdrojové kódy programu, takže je to testování zaměřené na programovou logiku
    - **Unit testy** (testování malých částí programů, jako jsou podprogramy nebo třídy)
    - **Integrační testy** (jsou testovány komponenty a jejich interakce na základě rozhraní)
    -
- Blackbox
    - Metoda testování bez znalosti kódu softwaru. Máme tedy k dispozici specifikaci softwaru a samotný software v podobě „černé skříňky", tzn. že se nemůžeme podívat dovnitř, jak funguje.
    - Řeší jiné typy chyb
        - nesprávné nebo zcela chybějící funkce
        - Chyby rozhraní
        - Chyby ve struktuře dat nebo externích databázích
        - Neočekávané chování
        - Chyby při inicializaci nebo ukončení
    - **Smoke test** (jestli to vůbec naběhne)
    - **Zátěžový test** (jestli se to sesype)
    - **Systémový test** (funkčnost v kontextu systému a interakce s jinými systémy)
    - **Hraniční testy** (vstupní data velmi blízko nebo na hranici akceptovatelnosti, v praxi je to totiž nejčastější zdroj problémů)
    - **Akceptační testy** (smoke test nového buildu a test zákazníkem po kompletním otestování)

Taky beta testing?

Vztah k sestavení produktu

Popsáno v jednotlivých metodách – různé typy v průběhu vývoje, před sestavením, po sestavení a před předáním, test zákazníkem

Měření software, produktové a procesní metriky, význam pro sledování kvality a řízení postupu

Metrika = způsob stanovení velikosti

Metriky software

- **Složitost, přehlednost**
  - počet možných cest skrz zdrojový kód
  - Fan-in / fan-out (afferent / efferent coupling) => stabilita
    - strukturální metrika, která měří poměr počtu modulů, které volají daný modul ku počtu modulů, které volá daný modul
  - Weighted Methods per Class
    - součet složitosti všech metod ve třídě
  - Lack of cohesion
    - nedostatek soudržnosti - jedna metoda dělá více funkcí (které se ve svém "smyslu" liší)
- **Velikost**
  - Počet Use Cases, funkčních bodů
  - Lines of Code
    - SLOC (Source Lines of Code),
    - DSLOC (Delivered Source Lines of Code),
- **Kvalita (nepřímé metriky)**
  - Pokrytí testy – kódu, požadavků
  - Charakteristika defektů – hustota, výskyt
  - Kvalita zdrojového kódu
- **Spolehlivost**
  - Střední doba mezi poruchami
  - dostupnost

Produktové a procesní metriky

Metriky produktu

- počet use case,
- počet podsystémů, modulů, tříd…,
- složitost modulů,
- počet řádků,
- datová velikost (ubuntu na 1 CD),
- počet odhalených chyb v jednotlivých modulech při testování,
- složitost dat modulu (funkční body),
- náklady na vývoj,
- člověkohodiny apod.

Metriky procesu

- Postup projektu

        - Rychlost vývoje
        - Change requesty a jejich zpracování,
        - Staff turnover (fluktuace zaměstnanců),
        - změny postupu/plánu
        - …
- Kvalita
    - Breakage = průměrná váha změny (LOC (Lines of Code) / CR (Change Rate))
    - Pracnost celkem, přepočtená na CR (Change Rate)
    - Množství chyb (procenta) odhalených před odesláním zákazníkovi.

Řízení postupu

- Plán měření
    - RUP template
- GQM (Goal Question Metric) přístup
    - Definice metrik, jejich význam a zpracování
- Sledování projektu a produktu
    - Automatické získávání a vyhodnocování
    - Sledování (management)
    - Korektivní akce

Význam pro sledování kvality a řízení postupu

- Lines of Code nic neznamená pro řízení kvality, ale třeba se dá odhadovat postup
- …