SSZ 2016 SWI

Návrh informačních systémů

NIS

Část: ASWI

Seznam otázek

1. Základní modely životního cyklu software, softwarový proces, metodika.

Životní cyklus – proces od zahájení vývoje až po vyřazení z provozu
Metodika - definuje jak má vypadat proces (fáze, aktivity, role, artefakty, milníky, ...)
Proces – systematická série akcí vedoucí k určitému výsledku

Softwarový proces

Je systematická série aktivit, souvisejících rolí a artefaktů vedoucí k vyšší pravděpodobnosti vytvoření kvalitního software.

- Výsledek – kvalitní software
- Mezi-výsledky – artefakty
- Členění – fáze, aktivity
- Činitelé – role

Aktivity v procesu:

- Technické – komunikace, plánování, modelování, konstrukce, nasazení
- Podpůrné – řízení, kontrola kvality, správa konfigurace, dokumentace

Role v procesu:

- Technické – analytik (konzultant), architekt (návrhář), vývojář, správce konfigurace, tester, databázista
- Manažerské – team leader, technický vedoucí projektu, šéf vývojářů, šéf projektů, CEO (výkonný ředitel obchodní společnosti)
- Podpůrné – poradce, kouč, lektor, uživatelská podpora, dokumentace

Artefakty

Mají účel, popis (dokumentaci), kontrakt, vlastnictví a jsou výsledek /vstupem aktivit => indikátor dosažení cíle

- Technické – specifikace, dokumentace, kód (data), testy, modely
- Komunikační - plán, specifikace
- Obchodní – plán, rozpočet, produkt

Varianty SW procesu

Společná snaha – snížení rizika chaotického postupu

- Řízené plánem – vodopádový model, V-model
- Řízené riziky – průzkumník/prototypování, spirálový model
- Řízené změnou – iterativní (RUP), agilní (SCRUM)

Základní modely životního cyklu software

Vodopádový model

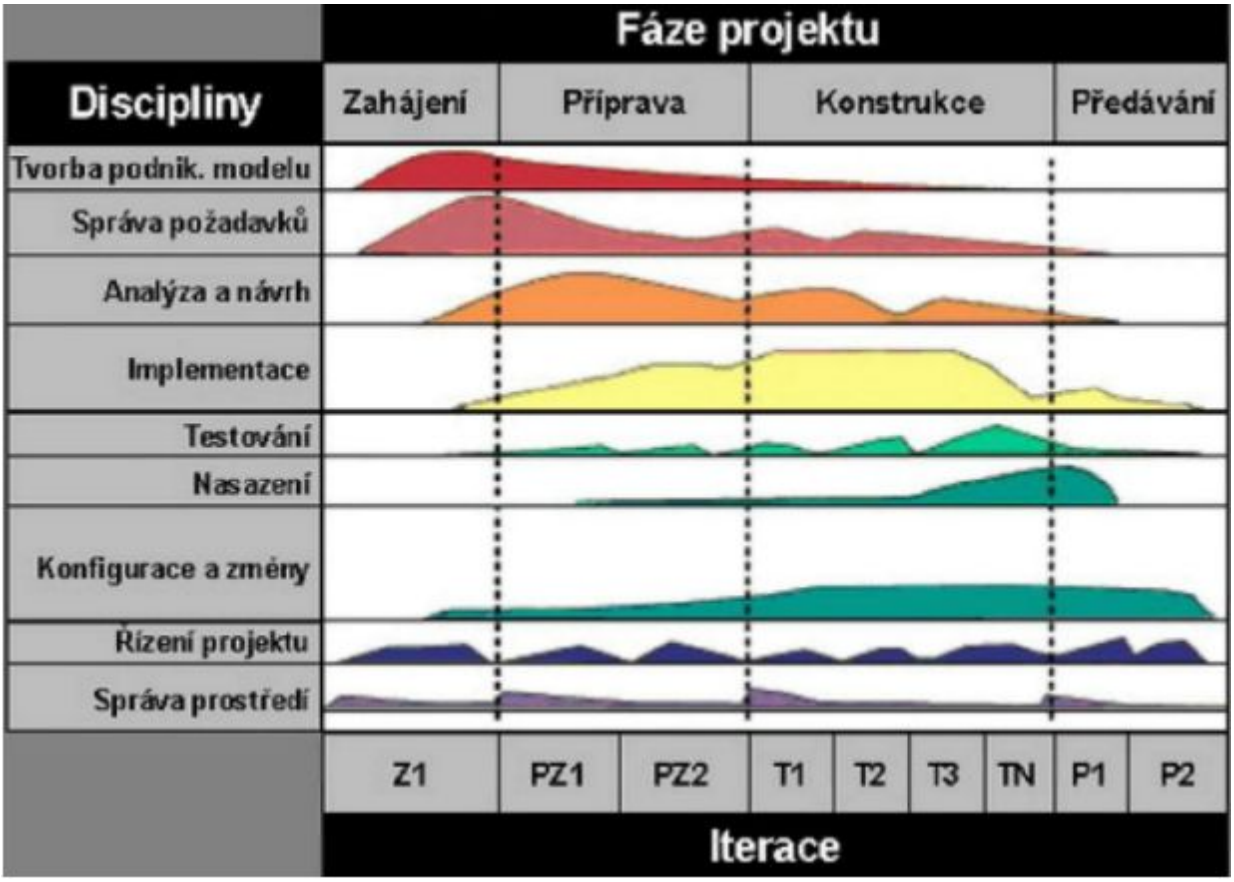
- Sekvenční přístup ke všem fázím modelu
- Vstoupit do další fáze až po ukončení předchozí
- Nevýhody – zpětná vazba od zákazníka až na konci projektu
- Použití pro malé projekty, výukové účely

Spirálový model

- Pokrývá nedostatky vodopádového modelu
- Postup do další fáze je možný až po analýze všech rizik a možných problémů (legislativa, marketing)
- Výsledkem každé fáze je prototyp (po použití se vždy zahodí)
- Model založen na iterativním přístupu (opakování analýzy všech rizik => snadná úprava požadavků)
- Model probíhá v několika krocích, které se neustále opakují, dokud není produkt hotový
- Navazování nových částí na již důkladně prověřený základ
- Vhodný pro větší projekty
- Životní cyklus podle spirálového modelu rozdělen do čtyř částí:
    - Plánování a určení cílů, alternativ, omezení
    - Vyhodnocování alternativ, identifikace a řešení rizik
    - Vývoj a verifikace další úrovně produktu
    - Plánování následující fází
- Po každé fázi následuje testování, hodnocení a předání dílčích výsledků

RUP

- Objektově orientovaný iterativní přístup k životnímu cyklu software
- Metodika se skládá ze čtyř základních fází a každá z nich obsahuje několik iterací
    - Zahájení – analytické činnosti, validace vize ze zákazníkem (1 – 2 iterace)
    - Projektování – analytické a designerské činnosti, ověřování prototypy (2+ iterace)
    - Konstrukce – designerské a programátorské činnosti, testování a ověřování (N iterací)
    - Nasazení - integrační a konzultační činnosti, ověřování provozem (1- 2 iterace)
- Po každé iteraci je k dispozici spustitelný kód
- Průběžná kontrola kvality produktu a modelování pomocí UML



Agilní metodiky

- Metodiky umožňující rychlý a hbitý vývoj softwaru
- Agilní – inkrementální vývoj (dodáván rychle, po malých částech)
- Co nejrychleji vyvinout, předložit zákazníkovi a podle zpětné vazby upravit

Manifest agilního programování:

- Jednotlivci a interakce před procesem a nástroji
- Fungující software před obsáhlou dokumentací
- Spolupráce se zákazníkem před vyjednáváním o smlouvě
- Reagování na změny před dodržováním plánu

XP (extrémní programování)

- Menší projekty a malé týmy, vyvíjející software podle zadání, které je nejasné nebo se rychle mění
- Principy a postup, které dotahuje do extrémů
    - Jestliže se osvědčují revize kódu, budeme neustále revidovat (párové programování)
    - Pokud se vyplácí testování, budeme testovat
    - Osvědčuje se návrh stane se součástí naší každodenní činnosti (refaktorizace)

SCRUM

- Každodenní setkání týmu – každý člen řekne co dělal, co bude dělat a na jaké problémy narazil
- Iterativní vývoj (období iterace se nazývá sprint, trvá 2- 4 týdny)
- Výsledkem sprintu je demo pro zákazníka (zpětná vazba), rychlejší reakce na změny
- Tři role – Product Owner (komunikace se zákazníkem), Scrum master (šéf vývoje), Scrum Team Member (člen týmu)

2. Sekvenční a iterativní přístup k vývoji software, výhody, nevýhody, důsledky; způsoby dodávky produktu.

Sekvenční přístup

- hlavní technické aktivity lineárně po sobě, např. vodopádový model
- Vztažené na celý produkt
- Naplánované pro celý projekt
- Oddělené meziprodukty

Výhody:

- Snadné pochopení
- Dobrá možnost řízení a sledování postupu řešení (milníky)
- Kladen důraz na dokumentaci – specifikace, design, analýza

Nevýhody:

- Vyžaduje mít na počátku přesně a úplně definované požadavky (uživatel často nedokáže stanovit předem)
- Zákazník vidí výslednou verzi až v závěrečných fázích řešení (nedostatky jsou odhaleny pozdě)
- Během vývoje se mohou měnit požadavky a výsledkem je, že dodaný software není to, co zákazník chtěl

Iterativní přístup

- Vývoj rozdělen na malé části, miniaturní úplně projekty s cca vodopádovým modelem (iterace)
- Cílem každé iterace je otestovaný a funkční produkt, i když je neúplný (iterační release)
- Iterativní vývoj má přírůstkový charakter – postupně zpřesňujeme a naplňujeme vizi
- V každé iteraci opakujeme stejné aktivity

Výhody:

- Menší časové úseky v dodávkách produktu => zvýšení úspěšnosti produktu díky zpětné vazbě
- Snížení rizik
- Snazší řízení změn na základě zpětné vazby uživatele
- Vyšší kvalita produktu

Nevýhody:

- Průběžné změny mohou způsobit porušení původní systémové struktury => náročnější údržba softwaru
- Vyžaduje přísnější management

Způsoby dodávky produktu

- Velký třesk – pro malé projekty, kde jsou jasné požadavky
- Přírůstkově – určení přírůstků, naplánování, postupné dodávky, zpětná vazba
- Evolučně – postupné zpřesňování, cyklus – určení cíle, dodávka, zpřesnění

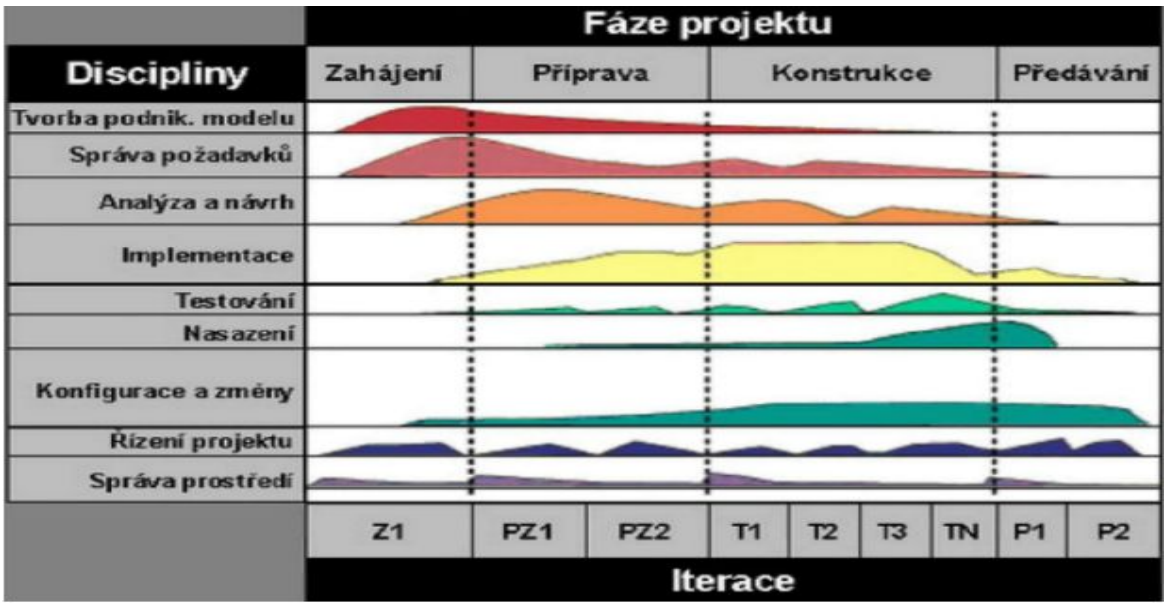
3. Základní charakteristiky iterativních a agilních metodik.

Přijde mi jako otázky kolem.. hlavně otázka 1.

Příkladem iterativní metodiky je například RUP a agilních metod SCRUM a extrémní programování. Ale nakopčím to sem...

RUP

- Objektově orientovaný iterativní přístup k životnímu cyklu software
- Metodika se skládá ze čtyř základních fází a každá z nich obsahuje několik iterací
  - Zahájení – analytické činnosti, validace vize ze zákazníkem (1 – 2 iterace)
  - Projektování – analytické a designerské činnosti, ověřování prototypy (2+ iterace)
  - Konstrukce – designerské a programátorské činnosti, testování a ověřování (N iterací)
  - Nasazení - integrační a konzultační činnosti, ověřování provozem (1- 2 iterace)
- Po každé iteraci je k dispozici spustitelný kód
- Průběžná kontrola kvality produktu a modelování pomocí UML



Agilní metodiky

- Metodiky umožňující rychlý a hbitý vývoj softwaru
- Agilní – inkrementální vývoj (dodáván rychle, po malých částech)
- Co nejrychleji vyvinout, předložit zákazníkovi a podle zpětné vazby upravit

Manifest agilního programování:

- Jednotlivci a interakce před procesem a nástroji
- Fungující software před obsáhlou dokumentací
- Spolupráce se zákazníkem před vyjednáváním o smlouvě
- Reagování na změny před dodržováním plánu

XP (extrémní programování)

- Menší projekty a malé týmy, vyvíjející software podle zadání, které je nejasné nebo se rychle mění
- Principy a postup, které dotahuje do extrémů
    - Jestliže se osvědčují revize kódu, budeme neustále revidovat (párové programování)
    - Pokud se vyplácí testování, budeme testovat
    - Osvědčuje se návrh stane se součástí naší každodenní činnosti (refaktorizace)

SCRUM

- Každodenní setkání týmu – každý člen řekne co dělal, co bude dělat a na jaké problémy narazil
- Iterativní vývoj (období iterace se nazývá sprint, trvá 2- 4 týdny)
- Výsledkem sprintu je demo pro zákazníka (zpětná vazba), rychlejší reakce na změny
- Tři role – Product Owner (komunikace se zákazníkem), Scrum master (šéf vývoje), Scrum Team Member (člen týmu)

4. Průběh iterace, její vlastnosti.

- Vývoj rozdělen na malé části, miniaturní úplně projekty s cca vodopádovým modelem (iterace)
- Cílem každé iterace je otestovaný a funkční produkt, i když je neúplný (iterační release)
- Iterativní vývoj má přírůstkový charakter – postupně zpřesňujeme a naplňujeme vizi
- V každé iteraci opakujeme stejné aktivity

Průběh iterace:

- Plánování cíle iterace (funkčnost)
- Doplnění / zpřesnění požadavků – plán projektu, vize a předchozí feedback
- Implementace přírůstku funkčností
- Integrace přírůstku – ověření, otestování
- Předání do provozu – validace se zákazníkem
- Zhodnocení

Počet a pravidla iterací:

- Dle charakteru projektu – rozsah projektu, velikost týmu
- Různé fáze projektu mají různý počet iterací, obvykle alespoň 3
- Pevné datum ukončení – plánováno nejpozději na začátku iterace
- Běžící iterace uzavřena změnám zvenčí – stabilita, změnové a projektové řízení

Délka iterace:

- Malá je lepší – blízký cíl, menší složitost a riziko, rychlá adaptace, produktivita
- Malé projekty – 1 až 4 týdny
- Velké projekty – 3 až 6 týdnů
- Agilní – SCRUM (30 dní), XP (1-2 týdny)
- Timeboxované iterace – délka iterace známa předem

Předání a zhodnocení iterace:

- Uzavření iterace
- „Customer Demo“ – předvedení, předání výsledku zákazníkovi
- Retrospektiva – co se dařilo (co zachovat), co se nedařilo a proč (co změnit), jak můžeme být příště lepší

5. Plánování, sledování a řízení iterativně vedeného softwarového produktu.

Plánování iterace:

- Na počátku projektu, na začátku každé iterace nebo mezi iteracemi
- V průběhu iterace se neplánuje!!!
- Iterace je miniaturní projekt
- Cílem je podmnožina požadavků na kompletní produkt
- Zachyceno v plánu iterace – backlog, bugtracker, dokument

Základní aspekty plánování:

- Nějaký plán nutný vždy – harmonogram, pevné body, přiřazení zdrojů
- Sledování plánu nutné vždy – kontrola postupu, reakce na změny
- Stupně volnosti plánování
    - Klasicky – měnitelné jsou čas, zdroje (cena) a pevné funkčnost
    - Agilně – čas, zdroje (cena) jsou pevné, funkčnost je měnitelná

Způsoby plánování:

- Adaptivní - detailně plánovat jen to na co máme data, postupně zpřesňujeme (globální pevné body plánu určeny předem) ▪ Prediktivní - naplánovat vše až do konce projektu (velká míra nejistoty)
- Plánování řízené riziky – vyhodnotit rizikové faktory projektu (designová, obchodní, legislativní, ...) a začneme s částmi funkčnosti s největší mírou rizika
- Plánování řízené prioritami klienta – zákazník určí funkčnost implementovanou v dané iteraci, reagujeme na potřeby zákazníka

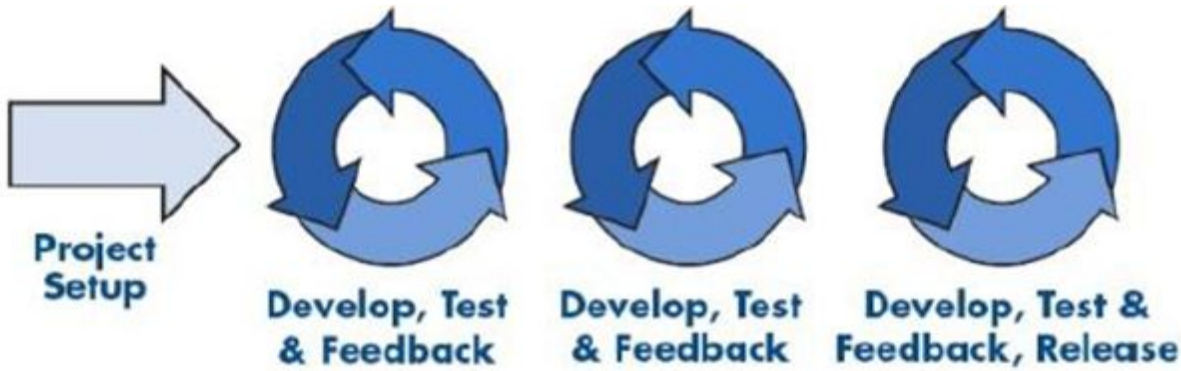
Jak funguje iterativní vývoj

„Když sekvenční postup funguje pro malé projekty s malou mírou neznáma, proč nerozbít velký projekt do řady malých?“

Miniaturní úplný projekt
- cca vodopádový model
- prolínání aktivit

Cíl: iterační release (interní či nasazený)
- produkt funkčně neúplný
- ale otestovaný a funkční

Opakovaný postup
- stále stejné aktivity (téměř)



Milníky vývoje – kritické místo v plánu (po stupních přesnosti a míře rizika)

LCO (Lifecycle Objectives)

- Srozumění s rozsahem, cenou, harmonogramem
- Souhlas s požadavky a jejich klíčovostí
- Navrhovaný postup vývoje souhlasí
- Rizika identifikována a řešení známo
- Artefakty – vize produktu, business case, seznam rizik, plán projektu, koncept technického řešení (architektura + prototypy)

LCA (Lifecycle Architecture)

- Vize a klíčové požadavky jsou stabilní
- Testy ověřili, že architektura řeší rizikové požadavky
- Jsou přesnější odhady pracnosti, na nich postavené plány
- Nástroje a postupy pro realizaci jsou v provozu
- Stakeholders – vize realizovatelná, spotřebované zdroje adekvátní
- Artefakty – vize produktu, seznam rizik a strategie řešení, popis architektury, specifikace požadavků, plán projektu

IOC (Initial Operational Capability)

- Je hotová „beta“ verze produktu
- Je hotová první verze plánu nasazení
- Implementace je dokumentovaná, existují testy
- Rozpracovaná uživatelská dokumentace
- Aktualizovány popisy návrhu, datového modelu, požadavků
- Artefakty – plán nasazení (první verze), testovací sady a reporty, architektura (aktualizovaná), popis implementace, uživatelská příručka

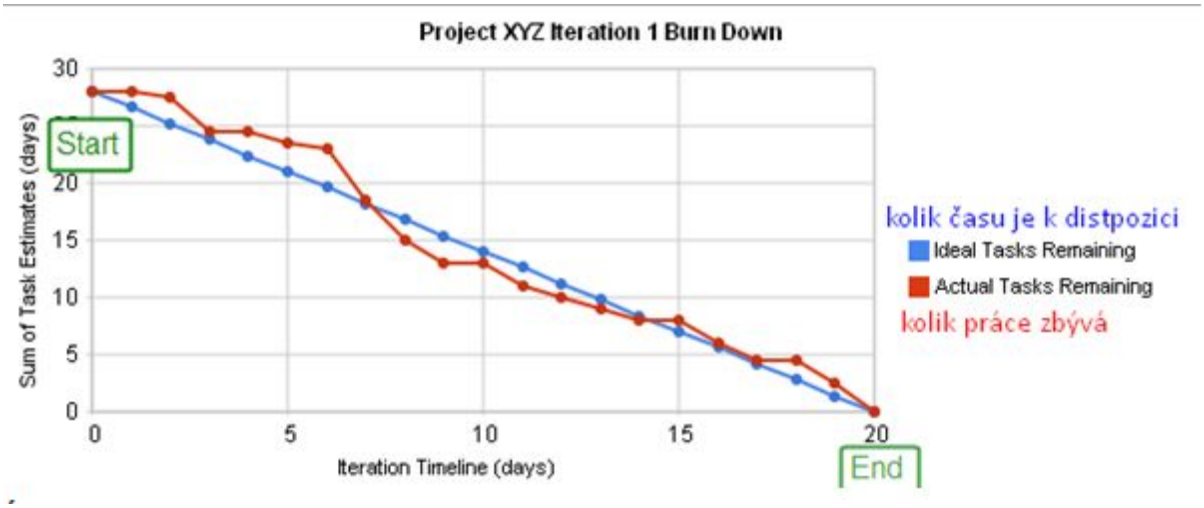
GA (General Availability)

- Uživatel spokojen s produktem
- Stakeholders jsou spokojeni s produktem
- Uvést produktu do rutinního provozu
- Support team
- Artefakty – release produktu, podpůrné materiály (uživatelská dokumentace), baseline kompletní konfigurace release

Sledování průběhu

- Nutnost. Důvody:
  - rozpoznat blížící se riziko
  - schopnost reagovat na změnu
- Project tracking and oversight
  - odhadovaný vs skutečně strávený čas

  Metody
  - reporting – analytické nástroje
  - veřejně přístupný plán („information radiator“)
  - komunikace – schůzky, XP role „Tracker“



Úpravy postupu

- Výchozí zkušenosti
  - plán není nedotknutelný
  - ani krátkodobé plány (iterace) se vždy nepovedou

Uvnitř iterace

  - opravdu nutné?
  - výjimečné stavy a jejich řešení

Mezi iteracemi

  - ideální chvíle pro reflexi a úpravy (plán, proces)
  - viz retrospektiva iterace

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

Požadavek

- schopnost nebo vlastnost, kterou má software mít, aby jej uživatel mohl používat k vyřešení problému nebo dosažení cíle, který vedl k zadání

Obsah a cíle požadavků:

- Popsat zadání tak, aby se z toho dalo vycházet pro implementaci
- Srozumitelné pro zákazníka/ analytiky, jednoznačnost a struktura pro návrháře, programátory, testery

Typy požadavků

podnikatelské požadavky
- formulují strategický rámec, říkají, proč organizace systém chce (čeho zavedením systému dosáhne)
- často samostatný dokument – tzv. charta projektu – popisuje vize a rozsah projektu

uživatelské požadavky
- popisují cíle uživatelů a úkoly, které musí být uživatelé schopni se systémem provést (způsob zápisu – případy užití, scénáře), např. převést peníze z účtu na účet
- mohou být v rozporu s podnikatelskými (pak je nutné komunikovat o cílech projektu a omezeních)

funkční požadavky
- popisují softwarovou funkcionalitu – úkol pro vývojáře, co mají naprogramovat

systémové požadavky
- celkové požadavky na systém složený z podsystémů (podsystémy mohou být softwarové i hardwarové, součástí systému jsou i lidé)
- odpovídají zejména definicím zdrojů a struktur

podnikatelská pravidla

firemní předpisy, státní nařízení, průmyslové standardy

parametrické požadavky (často nazývané také jako mimofunkční požadavky)

požadavky na výkonnost a kvalitativní parametry

omezení
- některé cesty při návrhu a vývoji nemohou být použity (např. existuje požadavek na konkrétní OS, nutnost integrace se stávajícím systémem, apod.)

Někdy se používá zjednodušené rozdělení

Funkční požadavky
- popisují funkce nebo služby, které jsou od systému očekávány

Mimofunkční požadavky
- netýkají se funkcí systému, ale vlastností jako je spolehlivost, čas odpovědi, obsazené místo na disku nebo v paměti, aj.

Formy popisu

- Textový popis – shopping list, backlog, story cards, strukturovaný text
- Grafické notace – případy užití, procesy
- Strukturovaný dokument – IEEE normy, RUP šablona (problém ... postihuje [koho] což vede k ... [důsledky] řešení bude ... [cílový stav])
- Implementace – prototyp, uživatelská příručka

Úrovně detailu v rámci procesu vývoje

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

Úrovně detailu a agilní metodiky

- Cíl – zachytit věci v danou chvíli nejpodstatnější (viz manifest agilního vývoje), detaily se dohodnou později
- S každou iterací zpřesnění – vize => features => epics => user story => task
    - **Theme (feature)** – seskupení souvisejících user stories (pracnost > 2 sprinty)
    - **Epic** – větší funkcionalita (větší user stories), rozpad na skupinu user stories, akceptační testy (pracnost > 1 sprint)
    - **User story** – jedna až dvě věty popisující z pohledu uživatele, co chce vykonat
    - **Task** – konkrétní zadání pro vývojáře, odvozeny z user story (pracnost max 1 den)

Backlog

- základní struktura pro zachycení požadavků, obsahuje epics + user stories (produkt), user stories + tasks (iterace)

Sprint

- základní jednotka pracnosti používaná v agilní metodice SCRUM a obvykle trvá 30 dní

7. Postup práce s požadavky, metody pro sběr a analýzu požadavků, použité modely a diagramy.

Obsah a cíle požadavků

- Popsat zadání tak, aby se z toho dalo vycházet pro implementaci
- Srozumitelné pro zákazníka/ analytiky, jednoznačnost a struktura pro návrháře, programátory, testery

Lidé v analýze

- Zákazník – externí, interní, doménový expert
- Stakeholder - zainteresovaný hráč (ředitel, investor, daňový poplatník, atd.)
- Analytik – zprostředkovatel mezi zákazníkem a programátory (komunikační schopnost, naslouchání pozorování)

Postup práce s požadavky

- Zjištění
- Analýza, vyjednávání – vytvoření konečných požadavků z potenciálních
- Dokumentace, zhodnocení
- Změnový management

Způsoby sběru požadavků

Interaktivní

- interview
  - předem připravený rozhovor, který vede moderátor (klade otázky, dává slovo)
  - nedoporučuje se více než2 hodiny
  - předem si připravit scénář, které okruhy se budou probírat, v jakém pořadí, scénářse snažit nenásilnědodržovat
- pozorování a práce s uživateli
  - pozorování prací u zákazníka (účast analytiků)
- dotazníky a průzkumy
  - vhodnými otázkami zjistíme od uživatelů, co potřebují
- prototypování
  - stačí na papír nebo skutečné programové prototypy

Neinteraktivní

- analýza existujícího systému
  - inspirujme se tím, jak funguje stávající systém
- studium dokumentace
- hlášení problémů (bugtracking)
- analýza trhu
- konkurenční systémy

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 požadavkůse 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 právěpotřebujeme
- 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 téměřautomatickou konverzi do kódu

Obrázky, prototypy 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 → mělo by se 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)

Modely

Doménový model

Doménový model se vytváří spolu s Use Case diagramem v počáteční fázi vývoje softwaru. Jedná se o formu **class diagramu**, tedy diagramu tříd. Základní entitou je třída. ?

Procesní model

Popisuje scénáře funkčnosti při složitém rozhodování, když je text nepřehledný. Notace může být vyjádřena např. UML diagramem aktivit.

Diagramy

Diagram tříd

- ukazuje statickou strukturu tříd v systému, tj. třídy, jejich vztahy (dědičnost, asociace, závislost), atributy a operace
- diagram je považován za statický proto, že struktura popisovaná diagramem platí v jakémkoli okamžiku běhu systému
- Vztahy
    - Agregace (závislost) - jeden prvek závisí na druhém
    - Asociace - propojení prvků
    - Dědičnost - specializace/generalizace

Diagram komponent

- kazuje organizaci a závislosti mezi komponentami nebo mezi komponentami a rozhraními komponent
- komponenty reprezentují dobře zapouzdřené prvky logické architektury
- komponenta zapouzdřuje implementaci a zveřejňuje množinu rozhraní
- komponenta může být implementována např. jedním nebo více spustitelnými soubory, zdrojovými texty nebo objektovými moduly, knihovnami, příkazovými soubory apod.

Diagram nasazení

- ukazuje fyzickou architekturu hardwaru a SW v systému, např. počítače, zařízení, jak jsou propojeny, jaké artefakty budou umístěny na kterých počítačích

Objektový diagram

- je variantou digramu tříd, diagram objektů ukazuje konkrétní instance a jejich vztahy (linky neboli propojení)
- ukazují možný vztah objektů v nějakém okamžiku běhu systému
- používá se poměrně zřídka, vhodné pro vysvětlení malých částí systému se složitými (zejména rekurzivními) vztahy

Diagram případů užití

- Případy užití se používají zejména pro popis kontextu systému a pro popis uživatelských požadavků.
- Diagram případů užití pomáhá nalézt hranice systému, vyhledat aktéry, nalézt případy užití a specifikovat případy užití.
- Komponenty: aktéři, případy užití, relace, hranice systému
- Aktér je uživatel nebo jiný systém, který analyzovaný systém používá
- Popis aktéra je pomocí názvu role ne jména
- Případ užití je sekvence akcí, které systém provádí v důsledku nějakého vnějšího podnětu a které vedou k výsledkům viditelným pro některého jeho uživatele.

Sekvenční diagram

- ukazuje dynamickou spolupráci mezi množinou objektů
- má časovou osu - plyne shora dolů v diagramu
- diagram ukazuje posloupnost zpráv zasílaných mezi objekty

Diagram spolupráce

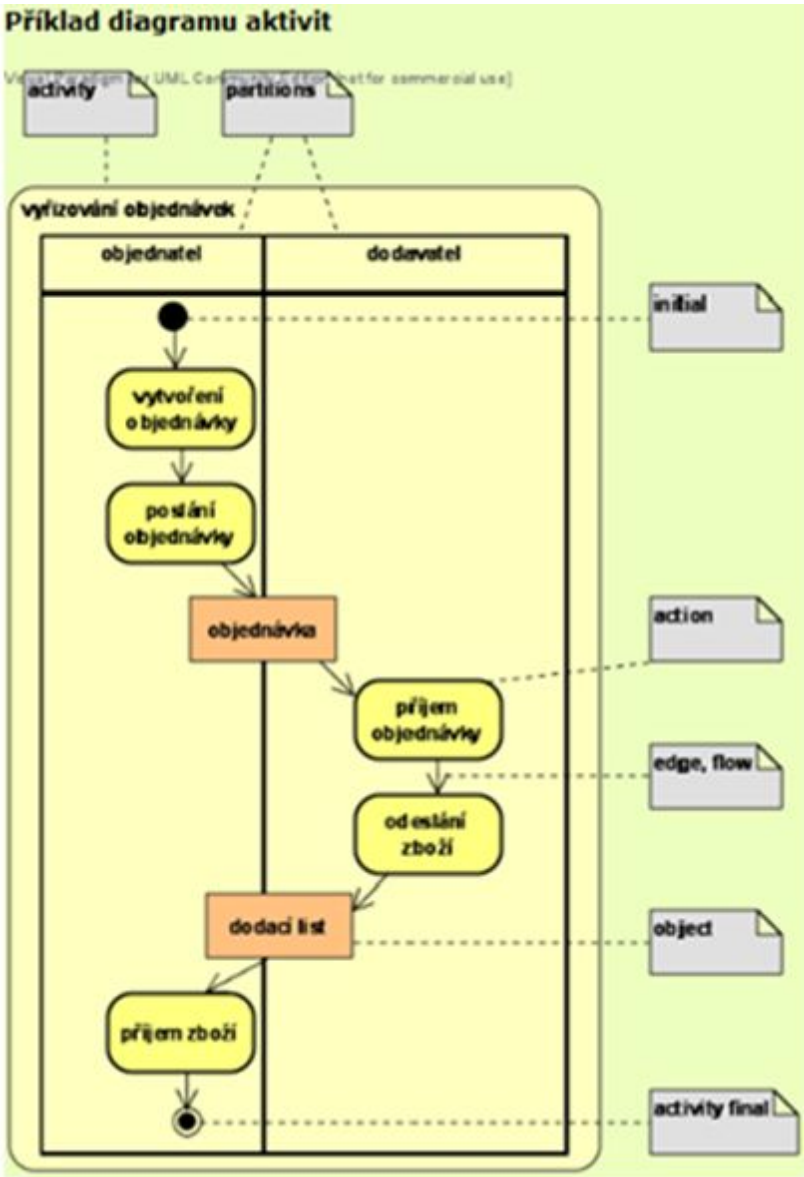
- podobně jako sekvenční diagram ukazuje výměnu zpráv
- na rozdíl od sekvenčního diagramu nemá časovou osu, zato ukazuje vztahy mezi objekty (linky)

Stavový diagram

- typicky se používá pro popis chování instance třídy
- ukazuje, jakými stavy mohou instance třídy procházet během svého životního cyklu a jaké události mohou způsobit přechod mezi stavy
- událost může být způsobena zasláním zprávy objektu, např. pokud uplyne specifikovaná doba nebo může přechod nastat po splnění nějaké podmínky
- prvky
    - stav = situace během života objektu, kdy objekt splňuje nějakou podmínku, provádí nějakou akci nebo čeká na událost
    - událost = výskyt stimulu, který může spustit přechod do jiného stavu
    - přechod = změna stavu způsobená událostí; nový stav závisí na původním stavu a na události

Diagram aktivit

- znázorňuje tok aktivit v rámci procesu
- umožňují modelovat paralelní chování, zpracování výjimek atd.
- založen na notaci Petriho sítí



8. 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
- Stanovuje základní kameny návrhu a základní směry vývoje a údržby

Součásti a formy popisu architektury

Logical view (logický pohled)

- Dělíme na menší celky – znuvupoužitelnost, údržba
- Horizontální vrstvy x vertikální domény
- Subsystémy, moduly, komponenty

Subsystémy:

- Funkčně soudržné a často vázané na jednoho aktéra
- Logická struktura subsystémů – balíky tříd, třídy a jejich vztahy (statický pohled)

Kdy začít členit do subsystémů:

- Standardní projekty – možno odhadnout předem
- Velké projekty – nahrubo předem, přesně během návrhu architektury
- Malé projekty – rozdělení na základě analýzy

Moduly (komunikace mezi moduly rozhraním)

- Základní stavební jednotka subsystémů
- Snaha o vícenásobnou použitelnost

Logické členění (balíky)

- Určeny pro logické členění – přehled, rozdělení implementace v týmu
- Balík tvoří skupina funkčně souvisejících tříd
- Hierarchické vnořování

Nalezení balíků:

- Rozdělení dopředu zřejmé – jednoduché anebo standardní aplikace, použití architektonických stylů
- Na základě objektového modelu – nutno vidět všechny třídy a vztahy

Physical view (fyzický pohled)

Komponenty

- Princip modularity a zapouzdření - aplikace jako Lego skládačka
- Komponenta je popsána svým rozhraním (poskytována a požadovaná rozhraní)
- Specifikace rozhraní a vlastností – manifest, deployment descriptor
- Interně se komponenta jeví jako black-box (implementace není důležitá)
- Technologie – CORBA, portlety, OSGi, ...

Process view (procesní pohled)

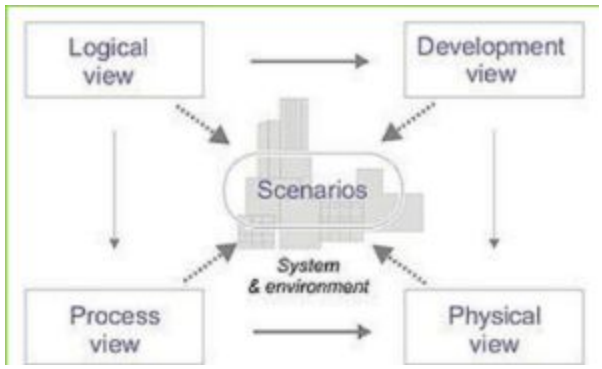
- Struktura paralelizace v implementaci
    - Procesy, vlákna, způsob synchronizace
    - Komunikace – synchronní / asynchronní
    - Propustnost, škálovatelnost, odolnost
    - Podpora (OS, jazyk, knihovny)

- Vazba na strukturu nasazení (distribuované systémy)
- Procesní model systému
    - Workflow - schéma provádění nějaké komplexnější činnosti (procesu), rozepsané na jednodušší činnosti a jejich vazby.
    - Alokace aktivit do modulů implementace
    - Synchronizace, předávání artefaktů

Development view (implementační pohled)

Konvence a politiky

- Obecná pravidla pro návrh v libovolné části aplikace, které musí všichni vývojáři dodržovat
- Návrhové vzory, správa paměti, synchronizace, transakce, defenzivní programování, lokalizace, dokumentace kódu



Architektonické styly

- Klient-server – tlustý/tenký klient
- 3-vrstvé a vícevrstvé – oddělení prezentace (prezentační, aplikační, datová), MVC
    - Prezentační vrstva
    - Aplikační vrstva (též Business Logic)
    - Datová vrstva
    - MVC vs 3-vrstvá
        - Model-view-controller má trojúhelníkovou topologii (ne třívrstvou) – pohled je obnovován (aktualizován) přímo modelem, na příkaz řadiče.
- Vrstvené – delegování požadavků na podřízené vrstvy
- SOA – servisně orientovaná architektura
- Pipes and filters („kolona“)
- Broker
- Peer-to-peer (P2P)

ISO/OSI - 7 vrstev (Fyzická-Linková-Síťová-Transportní-Spojová-Presentační-Aplikační)
TCP/IP - 4 vrstvy (Network Access=Ethernet/FDDI - Internet=IP - Transportní=TCP/UDP - Aplikační)

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

Konfigurační management a jeho součásti

Nutnost zabránit zmatkům při vývoji více verzí produktu ve více lidech

Konfigurační management (SCM)

- Proces identifikování a definování prvků systému
- Proces řízení změn těchto prvků během životního cyklu
- Proces zaznamenávání a oznamování stavu těchto prvků
- Proces ověřování úplnosti a správnosti těchto prvků

Prvek konfigurace – výsledek SW procesu

- Zdrojový soubor, dokument, model, knihovna, script, spustitelný soubor, testovací data, ...
- Atomický z hlediska identifikace změn
- Jednoznačně identifikovatelný - typ prvku (dokument, zdrojový text), označení projektu, název prvku, identifikátor verze

SW konfigurace – sestava prvků konfigurace reprezentující určitou podobu daného SW systému

- Jednoznačně identifikovatelná (např. verze X programu XY pro Linux)
- Obsahuje vše potřebné k jednoznačně opakovatelnému vytvoření dané verze produktu (build skripty, inicializační data, dokumentace)
- Konzistentní konfigurace – konfigurace, jejíž prvky jsou navzájem bezrozporné (např. zdrojové soubory jdou přeložit, knihovny přilinkovat)

Popis konfigurace:

- Množina prvků a jejich vzájemné vztahy tvoří strukturu produktu
- Vztahy:
    - Celek-část, master-dependent - určují strukturu a závislosti
    - Zdrojový-odvozený - určují způsob produkce (build produktu)

Role ve vývoji software

Určení a správa konfigurace

- Identifi kace prvků systému, přiřazení zodpovědnosti za správu
- Identifikace jednotlivých verzí prvků
- Kontrolované uvolňování (release) produktu
- Řízení změn produktu během vývoje

Zjišťování stavu systému

- Udržení informovanosti o změnách a stavů prvků
- Zaznamenávání stavu prvků konfigurace a požadavků na změny
- Poskytování informací o těchto stavech
- Statistiky a analýzy (vývoj oprav chyb) ▪ Správa sestavení (build) a koordinace

Správa sestavení (build) a koordinace

- Určování postupů a nástrojů pro tvorbu spustitelné verze produktu
- Ověřování úplnosti, konzistence a správnosti produktu
- Koordinace spolupráce vývojářů při zpracování, zveřejňování a sestavení změn

Role ve správě změn

Change Control Board

- Skupina členů projektu, která má zodpovědnost za změnové řízení
  - Vyhodnocování a schvalování hlášení problémů
  - Rozhodování o požadavcích na změny
  - Sledování hlášení a požadavků při jejich zpracování
  - Koordinace s vedením projektu
  - Složení: jedinec - vývojář, QA osoba; tým - technické i manažerské role

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

10. Správa verzí, způsoby verzování (revize a varianty, centralizované a distribuované), typické situace a vzory ve správě verzí, vazba na správu změn.

- Součást úlohy SCM „identifikace konfigurace“, tzn., aby prvek konfigurace mohl být ve správě
- SCM, musí být identifikovatelný, včetně svých podob.
- Účelem je udržení přehledu o podobách (verzích) prvků konfigurace – úložiště skladuje všechny verze

Prostředí pro verzování - úložiště

- Sdílený (centrální) datový prostor, kde jsou uloženy všechny prvky konfigurace projektu (ve všech verzích)
- Souborový systém ě dohodnutá pravidla
- Operace
  - Inicializace - vytvoření úložiště, naplnění boostrap verzí projektu
  - Check out – kopie prvku do lokálního pracovního prostoru
  - Check in (commit) – uložení změněných prvků do úložiště
  - Zjištění stavu – sledování změn v úložiště vs. Pracovní prostor
- Přístup k zamykání
  - Read-only pro všechny
  - Pesimistický – read-write kopie prvku jen pro pověřeného
  - Optimistický – read-write pro kohokoli, řešení konfliktů

Pracovní prostor - workspace

- Soukromý datový prostor, v němž je možno provádět změny prvků konfigurace, aniž by byla ovlivněna jejich podoba v úložišti
- Vývojářský – soukromý
- Integrační – sdílený

Možnosti verzování

- Jednotlivé prvky (verzování komponent) – konfigurace nemá verzi
- Celé konfigurace (úplné verzování) – verze konfigurace indukuje verze prvků
- Verze produktu

Druhy verzí

- Historická podoba – revize (Word 6.0)
- Alternativní podoba – varianta (Word pro Windows)

Určení konkrétní verze prvku

- Verzování podle stavu – identifikují se pouze prvky
- Verzování podle změn – identifikují se také změny prvků (výsledná verze vznikne aplikací změn)

Popis verze

- Extenzionální verzování – každá verze má jednoznačné ID (DOS 1.51)
  - 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 (OS = Win, arch=x64)
  - např. OS=DOS and UmiPostscript = YES

Informace o verzi

- Identifikátor verze (extenzionální) – jedinečnost, schémata (3.2.11.3, dle nástroje, marketingové jméno)
- Další meta-data prvku – datum a čas vytvoření, autor, stav prvku /konfigurace, předchůdce

Typické situace při správě verzí (větvení, značkování)

Codeline (vývojová větev)

- Série podob (verzí) množiny prvků konfigurace tak, jak se mění v čase
- Má daná pravidla práce s codeline – kam commitovat, přístupy, vytváření větví
- Vrchol codeline obsahuje nejčerstvější verzi (HEAD)
- Konfigurace může mít více codeline – vlastní projekt, knihovny třetích stran

Tag (label)

- označení konkrétní konfigurace symbolickým jménem

Baseline

- konzistentní konfigurace tvořící základ pro produkční verzi nebo další vývoj
- Stabilní – vytvořená, otestovaná a schválená managementem
- Změny prvků baseline jen podle schváleného postupu
- Při problémech návrh k baseline
- Význačené baseline – milníky projektu (interní release, alfa verze, beta verze, finální verze)
- Techniky – code freze (není povolena vůbec žádná změna kódu), stabilizační období

Paralelní práce na stejné konfiguraci

- Důvodem jsou velké úpravy, release, varianty, ...
- Cílem je vzájemná izolace paralelních prací, tak aby ukládané změny během nich neovlivnily ostatní (oddělení paralelních vývojových linií)
- Cena za izolaci od změn – řešení konfliktů

Kmen (trunk, master) – hlavní vývojová linie
Větev (branch) – paralelní vývojová linie
Spojení (merge) – sloučení změn na větvi do kmene (slučuje se delta od branch-off nebo posledního)
Delta (diff) – množina změn prvku konfigurace mezi dvěma po sobě následujícími verzemi
Changeset – delta + důvod změny
Diff – textový/binární rozdíl mezi verzemi
Patch – diff aplikovatelný na verzi

Nástroje pro správu verzí

Nástroje pro verzování

- ***Ruční verzování*** :-D → to přesně chceš dělat
- **Základní** - správa verzí souborů
  - Obvykle extenzionální verzování modulů
  - Centrální úložiště
  - Ukládání všech verzí v zapouzdřené úsporné formě
  - Příklad nástrojů: rcs, cvs, subversion...
- **Distribuované**
  - Více úložišť, synchronizace
  - Flexibilnější postupy
  - Příklad nástrojů: SVK, git, Mercurial
- **Pokročilé** - integrace do CASE

    - Obvykle kombinace extenzionálního a intenzionálního verzování
    - Automatická podpora pro check in/check out prvků z repository do nástrojů
    - Příklad nástrojů: ClearCase, Adele

Verzovací systémy

Centralizované

- **RCS**: revision control systém
    - Pesimistický přístup
    - 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 (inkrementální číslování revizí)
    - Souborová struktura
    - ***nejpoužívanější centralizovaný nástroj***

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
    - ***nejpoužívanější decentralizovaný nástroj***
- **Mercurial** – Netbeans, OpenJDK, Symbian OS
- **Bazaar** – Ubuntu

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)

Co nástroj má umět

- Operace s úložištěm – checkout, commit, add, remove, move, branch, merge, tag, import, …
- Verzování – data revize (klíčová slova), branch, merge, značkování ▪ Podpora týmu a procesu:
    - Vzdálený přístup
    - Konfigurovatelné zamykání a přístupová práva
    - Automatické oznamování
    - Spouštění scriptů při operacích
    - Integrace do IDE, řádkové a webové rozhraní

Prostředí pro verzování: úložiště

- Úložiště (databáze projektu, repository) = sdílený datový prostor, kde jsou uloženy všechny prvky konfigurace projektu
  - Zdrojové kódy
  - Knihovny (přeložené) a kód třetích stran
  - Konfigurační soubory, datové soubory
  - Scripty pro build, testování a instalace
  - Dokumentace, modely, prototypy
  - Odpadkový koš
- Řízený přístup (udržení konzistence)

Práce s úložištěm

- Základní operace
  - Inicializace - vytvoření úložiště, naplnění bootstrap verzí projektu
  - Check-out - kopie prvku do lokálního pracovního prostoru
  - Check in (commit) - uložení změněných prvků do úložiště
  - Zjišťování stavu - sledování změn z úložišti versus v pracovním prostoru
- Přístup k zamykání při check in/check out
  - Read-only pro všechny
  - Pesimistický: read-write kopie prvku jen pro pověřeného
  - Optimistický: read-write pro kohokoli, řešené konfliktů

Pracovní prostor

- Workspace = soukromý datový prostor, v němž je možno provádět změny prvků konfigurace, aniž by byla ovlivněna jejich oficiální podoba používaná ostatními vývojáři

Vazba na správu změn

- Vazba revize na ticket (hlášení problému) / change request (požadavek na změnu).
- Možnost požadavků na opravu / update konkrétních verzí

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

Typy požadavků na změny

- Požadavek na novou funkci / vlastnost
- Bug

Postup zpracování změny

- Vytvoření / přijetí – přidělení id
- Vyhodnocení – možná řešení, jejich dopady a odhad pracnosti, doplnění
- Rozhodnutí o
  - Způsob vyřízení – vyřešit, odmítnout, duplikát, odložit
  - Závažnost – kritická chyba, problém, vada na kráse, vylepšení
  - Priorita – vyřídit okamžitě, urgentní, vysoká, střední, nízká
- Zpracování – vygeneruje příslušné požadavky na změny
- Uzavření (nejprve všech požadavků na změny)
  - Build – ověření konzistence
  - Verzování – vytvoření nové baseline
  - Informovat zadavatele hlášení a další zájemce

Role

- Kdokoliv – přidání, aktualizace požadavků
- Change control manager – vyhodnocení a rozhodnutí
- Project manager – odhad pracnosti, přiřazení vývojaři
- Vývojář – zpracování požadavků
- Tester – otestování a validace vyřešeného požadavku

Change Controll Board

- Skupina členů projektu, která má zodpovědnost za změnové řízení
  - Vyhodnocování a schvalování hlášení problému
  - Rozhodování o požadavcích na změny
  - Sledování hlášení a požadavků při jejich zpracování
- Složení CCB – jedinec (vývojář, QA osoba), tým (technické i manažerské role)

Nástroje pro podporu řízení

Bug tracking (BT) systémy

- Evidence, archivace požadavků
- Vyhledávání
- Přehled reporty, grafy, statistiky
- Sledování stavu požadavku
- Realizace – emailové, webové, klientské
- Příklad – Mantis, Redmine, Bugzilla, Flyspray, JIRA

Vazba na správu verzí

- Vazba ticket (hlášení problému) / change request (požadavek na změnu) na verzi
- Vytvoření nové verze s opravou

12. Sestavení produktu –složky a vlastnosti, postup sestavení a jeho varianty, vztah k řízení kvality software, nástroje pro sestavení.

Řízení sestavení

- Aktivity provádějící transformaci zdrojových prvků konfigurace na odvozené – zejména sestavení celého produktu
- Cíle: vytvořit systematický a automatizovaný postup

Postup při vytváření sestavení (build proccess)

Build

- neboli sestavení je proces a výsledek vytvoření částečné nebo úplné podoby aplikace

Postup

- Příprava
- Check-out
- Preprocessing, překlad, linkování
- Nasazení
- Spuštění
- Testování
- Značkování, check-in
- Informování

Vlastnosti sestavení

- *Jedinečnost a identifikovatelnost*
- *Úplnost* – tvoří kompletní systém
- *Konzistence* – vzniklo ze správných verzí správných komponent
- *Opakovatelnost* – možnost opakovat build daného sestavení kdykoli v budoucnu
- *Dodržuje pravidla vývojové linie* – build odpovídající baseline
- Součásti prostředí pro sestavení: pravidla, skripty a daný HW

- Typy sestavení – čistý, úplný, přírůstkový
- Účel sestavení – soukromý, integrační, release
- Podpůrné aktivity – zkouška těsnosti, regresní testy, archivace prostředí, balení a distribuce

Obecný cíl: odchytit co nejdříve kdy a kde se to „rozbilo“

Typy sestavení

Soukromé sestavení

- Cíl: ověřit si konzistenci konfigurace – produkt lze sestavit pro mnou provedených změnách, předh check-in (problémy řeším já x všichni)
- Postup: sestavit produkt v soukromém prostoru
- Urychlení průběhu – použit inkrementální sestavení tam kde je to vhodné, vynechat postupy pro balení, vkládání info o verzi, pomoci si sdílením odvozených prvků (shared version cache)

Integrační sestavení

- Ověřit, že celý produkt lze sestavit
- Sestavuje se **centrálně, automatizovaným a opakovatelným procesem (celý produkt vč. závislostí)**
    - postup co nejpodobnější sestavení pro release
    - maximální automatizace – typicky běží přes noc
    - mechanismy zaznamenání chyb a informování o nich
    - úspěšné sestavení může být označkováno ve verzovacím systému
- Sestavuje se vždy „na zelené louce“
- Emailové notifikace – začátek, konec, výsledek + detaily
- Úplné sestavení může být označkováno ve verzovacím systému

Release sestavení

- Pro interního zákazníka(QA) nebo pro trh/externího zákazníka
- Náležitosti release:
    - revize/verze konfigurace použité pro sestavení
    - datum vytvoření
    - identifikátor sestavení
    - další metadata: zodpovědná osoba, zdrojová znakča konfigurace (zverzovacího systému), jakými prošlo testy (a výsledky), cesta k logům překladu (a testů)
    - „marketingová verze“ např Open cms 7.5

Nejlepší praktiky: SCM + QA

Zkouška těsnosti (Smoke Test)

- Ověření, že sestavení vytvořilo funkční produkt
- Samotné sestavení toto nezaručuje
- Vytvoření testů, které ověřují jen základní funkčnost
- Odchycení nejkřiklavějších chyb

Regresní testy

- Zajistit, aby nové funkce a vylepšení nesnižovaly kvalitu již hotového kódu => vymyslet jak vytvářet vhodné testy
- Ověřit build produktu pomocí testů, kterými již dříve prošel
- Produkt selhal => napsat test, který to dokáže => přidat jej do sady regresních testů

Daily Build + Smoke test

- Integrační sestavení + zkouška těsnosti
- Pravidelně 1x denně
- Výsledky okamžitě známy a reflektovány
- Zvladatelné množství změn během denních check-in
- Lepší morálka týmu
- Trocha discipliny, trocha automatizace

Nástroje

- Scriptovací – shell, perl, python, php
- Buildovací – make, ant, maven
- Verifikace sestavení – xUnit (např jUnit), testovací roboti

13. Způsoby prevence chyb v software, použití metrik kvality, oponentury.

Způsoby prevence chyb

- Cíl: **zabránit vzniku a dalšímu šíření chyby**
- Využití Racionální proces a best practices
- Kontroly a měření meziproduktů (častější v úvodních fázích)+
  - **Automatizované testy**
    - Základní kontrola kvality kódu → typicky **unit** testy
  - **Prověření meziproduktu nezávislým oponentem** (dříve než se z něj začne vycházet v další práci)
  - **Technická oponentura** a podobné techniky (Faganovská inspekce)
    - Viz oponentury.
  - **Párové programování, refactoring**
    - ***Párové programování***
      - Metafora řidič (udává směr, vysvětluje, naslouchá) + navigátor (dohledává, kontroluje, pomáhá) - svědomí páru (společný cíl, plné nasazení, komunikace)
    - ***Refactoring***
      - Změna interní struktury software, která jej činí srozumitelnějším a snáze upravitelným, aniž by změnila jeho vnější chování
      - Detekce "zapáchajícího" kódu
      - Změna designu, oprava
  - **Strukturované procházení**
    - Podobné Faganovské inspekci, menší důraz na formálnost
  - **Peer review**
    - Kontrola nezaujatým čtenářem
    - Autor prochází kód a vysvětluje
    - Kolega hledá problémy a komentuje
  - **Code review**
- Měření
  - Kvantitativní ukazatele pomáhají najít slabiny kvality
  - Přesnost a dokazatelnost, možnost statistik
  - GQM přístup, FURPS

Detekční a opravné techniky

- Cíl: najít a opravit již existující chybu
- Testování a ladění (typické v koncových fázích, tzv. výstupní kontrola)

Psaní čistého kódu

- snižuje riziko "ukrytí" zákeřných programových chyb už v prvotní fázi psaní kódu
- podporuje efektivnější ladění

Refaktorizace kódu

= změna struktury kódu, která nemá vliv na jeho celkovou funkčnost (přejmenování proměnné/metody, vyjmutí kódu do samostatné metody,..)

- snížení složitosti kódu
- zpřehlednění

Revize kódu (Code Review)

- Prováděna ideálně během nebo těsně po vlastní implementaci
- Revize kódu je jedním ze základních aspektů párového programování (Pair Programming)
- Efektivnost závisí na zkušenostech vývojáře a “revizora”
- Začínající vývojář (revizorem zkušený pracovník) X zkušený vývojář (revizorem posluchač získavající zkušenosti a praktické rady)
- Jsou různé techniky revizí kódu, liší se svojí formálností, obsazením revizního týmu a způsobem evidence nalezených defektů

Statická analýza (kontrola) kódu

= analýza kódu, která je prováděna bez nutnosti spouštění programu – soubor preventivních technik. Ověření formální správnosti + dodržování pravidel + metriky

- Nástroje: překladač a jeho hlášení
- Postupy: párové programování

Automatizované testování

- Dynamická analýza kódu
- Podle **TDD** (**Test-Driven Development**) by mělo předcházet vlastní implementaci
- Testy jsou indicatory chybových stavů
- **Unit testy**
    - Volání metod instance testované třídy s určitými vstupy a validace výstupů
    - Pokud výstupy neodpovídají předpokladům, test selže a je nutno hledat chybu v implementaci.
    - Musí být správně cílené na problémové situace
    - White-box testování – test “vidí” do implmentace
        - Pokrytí cest provádění
    - Black-box testování – testovaný kód není pro test přístupný
        - zosobňuje naivní přístup klientské strany, který může přinést nečekané způsoby volání
- Integrační a systémové testy
    - Validují interakci vice objektů a funkcionalitu větších celků
- Testy by měly být vzájemně nezávislé

14. 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 program => 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)
  - **Regresní testy** (opravím chybu a napíšu na to test ⇒ už se to nikdy neposere)
- 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 tes**t (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)
- **Usability testing** – testování uživatelem, hodnocení přívětivosti, intuitivnosti, …

Beta testing sem nespadá – může se skládat z akceptačních testů, usability testů atd. poté, co se dosáhlo beta milestonu a produkt se v této fázi testuje

Vztah k sestavení produktu

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

15. Měření software, produktové a procesní metriky, význam metrik pro sledování kvality a řízení projektu.

Proč měřit?

- Kvantitativní ukazatele – kvalita, přesnost, efektivita
- Dávají přehled a kontrolu nad projektem – plán, kvalita, splnění požadavků
- Kalibrují odhady
- Výhody
  - přesnost, možnosti statistik, vizuální prezentace

Jak měřit

- *Top-down*
  - Definovat cíl měření → zvolit metriky
- *Goal-Question-Metric*
  - Goal – problém + cíl měřícího programu – zlepšit spravedlivost v oceňování práce na projektu
  - Question – měřené objekty a způsob měření – kolik práce odvádí jednotliví členové týmu
  - *Metric* – konkretizují získávaná data – počet řádek v svn; váha uzavřených tasků v bugtrackeru
- *Bottom-up*

Metrika = způsob stanovení velikosti, měřitelná charakteristika nějaké entity, např. produkt, proces

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 dělái 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)
  - Porytí testy – kódu, požadavků
  - Charakteristika defektů – hustota, výskyt
  - Kvalita zdrojového kódu
- **Spolehlivost**
  - Střední doba mezi poruchami
  - dostupnost

Produktové a procesní metriky - viz předchozí otázka

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
- Je nutné sledovat kvalitu a upravovat vůči ní proces vzhledem k nákladům na zdroje, čas. Navíc pokud mineme chybu a vyjde do produkce, kromě řádově vyšších nákladů můžeme poškodit jméno společnosti