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

* Výsledek – kvalitní software
* Mezivýsledky – 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 hotov
* 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)

# 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

# Základní charakteristiky iterativních a agilních metodik.

* 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

## Globální řízení iterativního projektu

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

* LCO – definování vize produktu
* LCA – určení způsobu řešení (architektura technického řešení), ověření (modely, prototypy, testy)
* IOC – vyrobit efektivně řešení (beta verze, všechny funkčnosti), jednotkové a funkční testy
* GA – uvést produkt do rutinního provozu (krabice s produktem), podpora

**Charakter iterací dle fáze - z**ákladní schéma pevné, mění se činnosti a artefakty

* Zahájení – analytické činnosti, validace vize zákazníkem (1 – 2 iterace)
* Projektování – analytické a designerské činnosti, ověřování prototypy, implementace (2+ iterací)
* Konstrukce – designerské a programátorské činnosti, změnové řízení, testování a ověřování (N iterací)
* Nasazení – integrační a konzultační činnosti, ověřování provozem, náběh uživatelské podpory (1 – 2 iterací)

# Vlastnosti iterace a její průběh

* 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 (develop, test, feedback)

**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

# Plánování a řízení iterativně vedeného softwarového projektu.

**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

**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

**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

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

**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

* Bussines requirements – z vize a rozsahu projektu (náklady, smluvní záležitosti)
* User requirements – funkční požadavky
* Bussines rules, constraint – fakta o fungování „business logiky“
* Extra-functional – mimofunkční požadavky (odezva, spolehlivost, …), FURPS
* Systémové požadavky
* Právní požadavky (různé právní systémy dvou zemí)

## 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

# Postupy pro sběr požadavků

**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, pozorování a práce s uživateli, dotazníky, průzkumy, prototypování
* Neinteraktivní – studium dokumentace, hlášení problémů (bugtracking), analýza trhu, konkurenční systémy

# Analýza požadavků a tvorba objektového návrhu – postup, použité modely a diagramy.

**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 (textová, případy užití)

## Tvorba objektového návrhu

**Datové modelování**

* Doménový model
* Logický (konceptuální) datový model
* Fyzický datový model

**Doménový model**

* Popis struktury problémové oblasti – základní abstrakce v oblasti, názvy, vlastnosti a vzájemné vztahy
* Východiskem je glosář – seznam důležitých pojmů (slovník, prevence nedorozumění)
* Model – doménové objekty (**diagram tříd**)

**Doménové objekty / třídy**

* Doménové objekty / třídy – „věci“ vyskytující se v problémové doméně, klíčové pro fungování systému, systém udržuje informace
* Podstatné aspekty – názvy tříd, základní obry, vztahy mezi třídami (asociace, kardinality)



**Jak najít doménové objekty**

* Doménová analýza – konzultace, doménový expert, části systému podstatné z jejich pohledu
* Pomůže – obrázek, rozhovor s uživatelem, pozorování práce

**Fyzický rozsah systémy**

* Vztahy produktu k prostředí – porozumění run-time a fyzickému prostředí
* UML – **model nazazení**



# Architektura softwarových systémů, význam a součásti architektury, formy popisu 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

Na základě případů užití – potlačit nepodstatné PU a pro zbylé zkusit vyjmout klíčovou třídu



## Architektonické styly

* Klient-server – tlustý/tenký klient
* 3-vrstvé a vícevrstvé – oddělení prezentace (prezentační, aplikační, datová), MVC
* Vrstvené – delegování požadavků na podřízené vrstvy
* SOA – servisně orientovaná architektura
* Broker

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

# Správa verzí, možnosti verzování, typické situace při správě verzí (větvení, značkování), nástroje pro správu 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)
* Intenzionální verzování – verze je popsána souborem atributů (OS = Win, arch=x64)

**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
* Základní (rcs, cvs, subversion) – centrální úložiště, extenzionální verzování
* Distribuované (git, hg, bzr, …) – více úložišť, synchronizace
* Pokročilé (ClearCase, Adele) – integrace do CASE, kombinace extenzionálního a intenzionálního verzování

**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

## Vazba na správu změn

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

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

## 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
* 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

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

## Sestavení produktu

* Sestavení (build) – proces a výsledek vytvoření částečné nebo úplné podoby aplikace
* Aktivity provádějící transformaci zdrojových prvků konfigurace na odvozené (sestavení celého produktu)
* Cílem je vytvořit systematický a automatizovaný postup

**Postup při vytváření sestavení :**

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

**Vlastnosti sestavení:**

* Jedinečnost a identifikovatelnost – čitelný jednoznačný (automaticky generovaný) identifikátor
* Úplnost – obsahuje všechny komponenty
* Konzistence – vzniklo se správných verzí komponent
* Opakovatelnost – build se dá v budoucnu kdykoliv opakovat se stejným výsledkem
* Dodržení pravidel vývojové linie – build odpovídající baseline, zejména release má striktní pravidla

**Typy sestavení:**

* Dle vstupu – čistý, úplný, přírůstkový (přeloží se jen změněné soubory)
* Dle účelu – soukromý, integrační (oficiální), release build

## Postup sestavení a jeho varianty

**Základní postupy pro sestavení:**

* Soukromé sestavení:
* Pro vývojáře, aby si ověřil, že svojí změnou nic nerozbil
* Co nejvíce podobné oficiálnímu sestavení (obsahuje všechny závislé části a prostor je podobný)
* Urychlení průběhu – inkrementální sestavení, kde je to vhodné, sdílení odvozených prvků, vynechání některých částí
* Integrační sestavení:
* Cílem je ověřit, že produkt jde sestavit (soukromý build nestačí a úplný trvá moc dlouho a někdy ho nemůže provádět vývojář)
* Postup – celý produkt sestaven centrálně automatizovaným a opakovatelným procesem (nejpodobnější sestavení pro release)
* Maximální automatizace celého procesu
* Release sestavení:
* Význačné integrační sestavení, které se dodá zákazníkovi (interní, QA)
* Náležitosti – verze (kódu, knihoven), datum identifikátor sestavení, další metadata, marketingové označení

**Podpůrné aktivity:**

* Kusovník – kompletní seznam prvků sestavení, umožní kdykoliv reprodukovat stejné sestavení
* Archivace prostředí – správa verzí objektů, které nejsou v úložišti (nástroje, HW, prostředí, platformy, …)
* Zkouška těsnosti (Smoke Test)
* Ověřit, že sestavení vytvořilo funkční produkt (samotné sestavení nezaručí)
* Provedou se rychlé, základní testy (kompletní by trvalo moc dlouho)
* Odchytí se nejkřiklavější chyby, odpustí drobnosti
* Spouštěno při každém sestavení
* Regresní testy:
* Cílem je ověřit, že změny nebo přidané vlastnosti neměly žádný vliv na stávající funkčnost (opakem progresní testy – nové vlastnosti)
* Ověření sestavení pomocí testů, kterými již dříve prošlo
* Zdroje testů – chyby objevené QA, validací, zákazníkem => pro chybu napsat test
* Balení a distribuce

**Nejlepší praktiky pro iterativní a přírůstkový vývoj:**

* Jednotkové testy (unit test) – zejména pro refactoring
* Pravidla pro codeline aktivního vývoje
* Daily build and smoke test – denní sestavení a zkouška těsnosti
* Regresní testy
* Continuous build – soustavné sestavování

**Daily build and Smoke Test:**

* Integrační sestavení + smoke test, probíhají alespoň 1x denně (většinou v noci)
* Jsou pravidelně vidět výsledky, rychlá hlášení problém (při zvladatelném množství změn)
* Zvladatelné množství oprav, lepší morálka týmu

**Continuous Integration:**

* Daily build dotaženy do extrému – sestavujeme neustále, pořád dokola (automatizace)

## Nástroje pro sestavení

* Skriptovací – shell, perl, python, php, …
* **Buildovací**
* Make – sestavení projektu na základě popisu závislosti typu zdrojový-odvozený, makefile (definice pravidel)
* Maven - deklarativní build, pevně daná struktura projektu)
* Ant, Hudson, …