

► Vývoj software má N rozměrů



► ... běžná aktivita v informační společnosti

Na zakázku
Interní projekt
Krabicový software
„Pro radost“

Closed source
Open source
(+ reuse)

Utilita
Systémová
komponenta
Mission-critical
software

Na zelené louce (green field)
Rozvoj existujícího produktu
Integrační projekt

Komerční zákazník
Státní sféra
Vertikály (utility, banky,
telco, ...)

13

► Softwarový proces



► Proces: *systematická série akcí vedoucí k určitému výsledku*

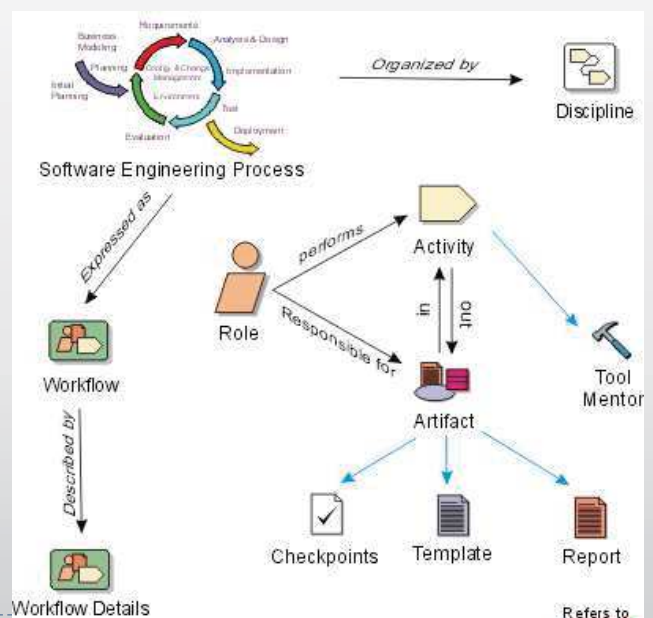
[Random House Unabridged Dictionary, 2006]

► Softwarový

- výsledek = kvalitní software
- členění: fáze, **aktivity**
- mezivýsledky: **artefakty**
- činitelé: **role**

► Meta-proces, ŽC

- varianty uspořádání aktivit, produktů



22

▶ Průběh iterace



- ▶ Plánování cíle iterace
 - ▶ zejména funkčnost
- ▶ Doplnění / zpřesnění požadavků
 - ▶ základ: plán projektu, vize, předchozí feedback
- ▶ (Úprava návrhu)
- ▶ Implementace přírůstku funkčností
- ▶ Integrace přírůstku
 - ▶ ověření, otestování
- ▶ (Předání do provozu)
 - ▶ validace zákazníkem
- ▶ Zhodnocení



▶ 5

▶ Globální plánování: milníky



- ▶ Cíl: eliminovat momentálně největší riziko
- ▶ **Barry Boehm (1996): Anchoring the Software Process**
- ▶ **LCO (Lifecycle Objectives)**
 - ▶ definování terče – Vize produktu
- ▶ **LCA (Lifecycle Architecture)**
 - ▶ určení způsobu řešení – Architektura technického řešení
 - ▶ ověření – modely, technické prototypy, testy (executable)
- ▶ **IOC (Initial Operational Capability)**
 - ▶ schopnost efektivně „vyrobit“ řešení – beta verze, all features
 - ▶ unit a funkční testy
- ▶ **GA (General Availability)**
 - ▶ uvést produkt do rutinního provozu – „krabice“ s produktem, website launch, raut :-)
 - ▶ support team v provozu

▶ 13

▶ Milník = kdy jsou cíle fáze dosaženy

▶ LCO

- ▶ srozumění s rozsahem, cenou, harmonogramem
- ▶ viz **Boehm: Anchoring the Software Process**



▶ Artefakty

- ▶ Vize produktu, Business case
- ▶ Seznam rizik a strategie jejich řešení
- ▶ Slovník pojmů a přehled klíčových požadavků
- ▶ Koncept technického řešení (architektura + prototypy)
- ▶ Plán projektu
- ▶ Popis procesu a infrastruktury

▶ Postup práce s požadavky



▶ Reqts development

- ▶ Elicit
- ▶ Analyze, Negotiate
 - potential → stable requirements
- ▶ Document
- ▶ Review
 - baselined requirements

▶ Reqts management

- ▶ Change management

► Typy požadavků



- Business reqts
 - Vize a rozsah projektu
- User (funkční) reqts
- Business rules, Constraints
- Extra-functional
 - vlastnosti
- System reqts
- Contractual, legal, ...

Příklad:
business rules

► Vize produktu: kostra



- Popis **problému a účelu**
 - smysl a účel cílového produktu
 - obchodní příležitost, důvod ekonomické návratnosti
- Přehled **stakeholders**
 - kdo jsou zájemci o systém, typy uživatelů
 - (potenciální konkurence)
- Přehled očekávaných **schopností** a funkcí produktu
 - popis, kvalitativní charakteristiky, priority
 - stručný výčet bez detailů
- **Omezení, standardy, závislosti**
 - vztahující se k projektu
- (Rámec **plánu** projektu)
 - časový rozsah, plánované verze / vydání

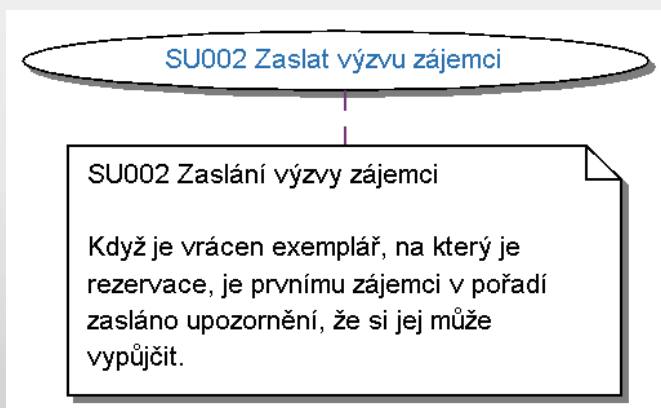
Příklad: Wiegiers

► Základní popis případu užití



... ve fázi shromažďování požadavků:
základní popis dané funkce aplikace

- Název (+ ID)
- Stručný popis – 1 věta
- Případně
 - Základní kroky postupu pro klíčové PU
 - Odkazy na zdrojové informace

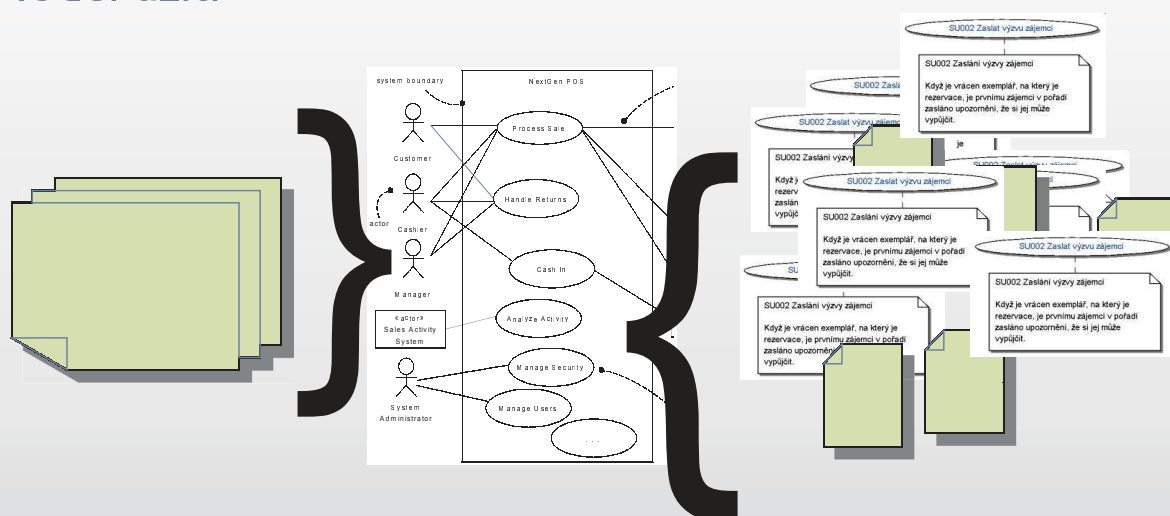


► 22

► Obrysy požadavků s případy užití



- Charakteristiky aktérů
- Model užití



- Jeden dokument (diagram jako „obsah“) nebo informace v UML nástroji

► 24

► Forma: User Story



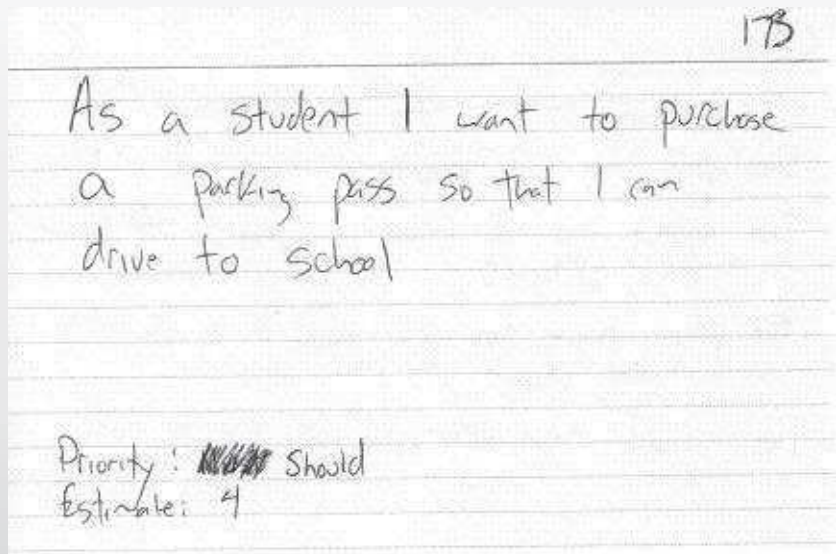
► Co uživatel od systému očekává a proč

► Obsah

- název
- stručný popis
- důležitost

► Způsob zápisu

- karta
- položka v ALM nástroji



<http://www.agileconnection.com/article/how-do-i-write-requirements-using-stories-and-acceptance-criteria-part-one>
<http://www.agilemodeling.com/artifacts/userStory.htm>

► 27

► Obrysy požadavků agilně: Backlog

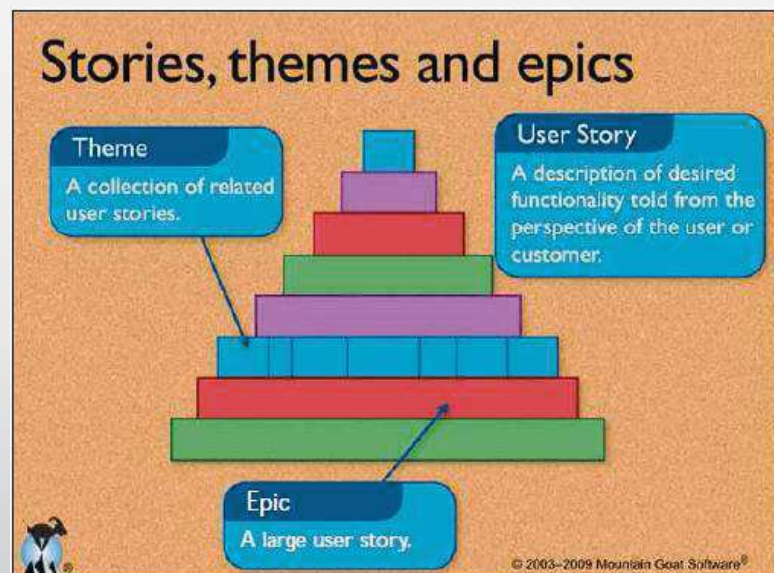


► Product Backlog = základní struktura

- obsahuje epics, stories
- just-in-time zpřesňování (příští iterace)

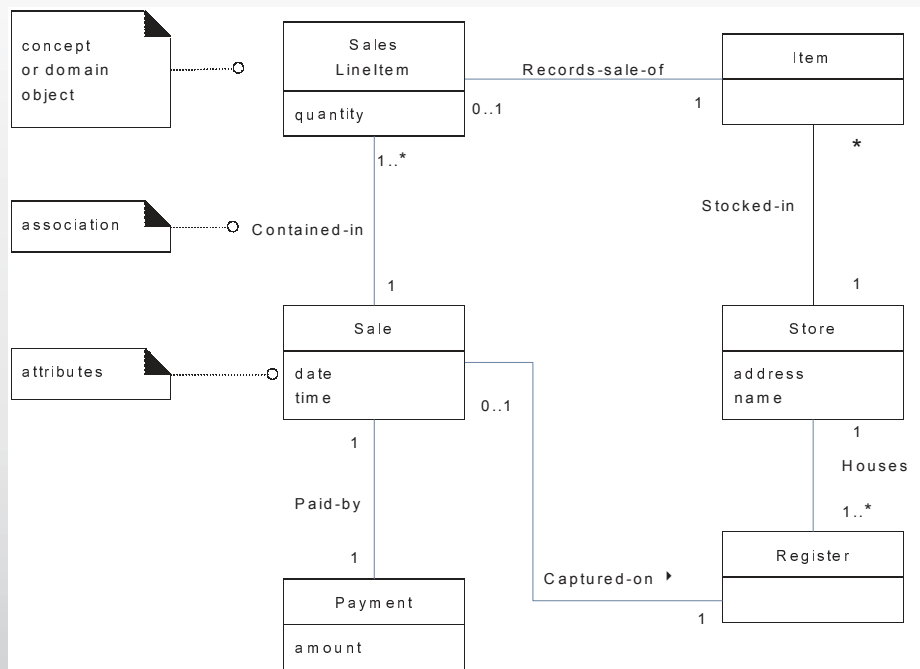
► Nejen požadavky

- viz Plánování



► 29

► Obrysy základních struktur s UML: doménový diagram



► 34

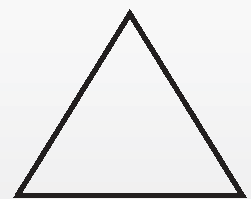
► Stupně volnosti při plánování



► Klasicky: čas, zdroje (cena), kvalita

- obtížně měnitelné, odhadované
- kvalita obtížně říditelná

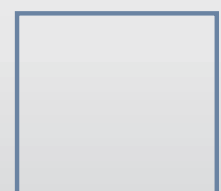
- typický požadavek: „bude to v termínu, s daným rozpočtem, samozřejmě v bezchybné kvalitě“
- typická realita: „you get crappy SW late“



► Agilně: +funkčnost

- nejlepší faktor pro řízení projektu
 - první tři pevné, funkčnost nejsnáze měnitelná
- vhodná granularita → snadné a přesné odhady

*Cheap. Fast. Good.
Choose any two.*



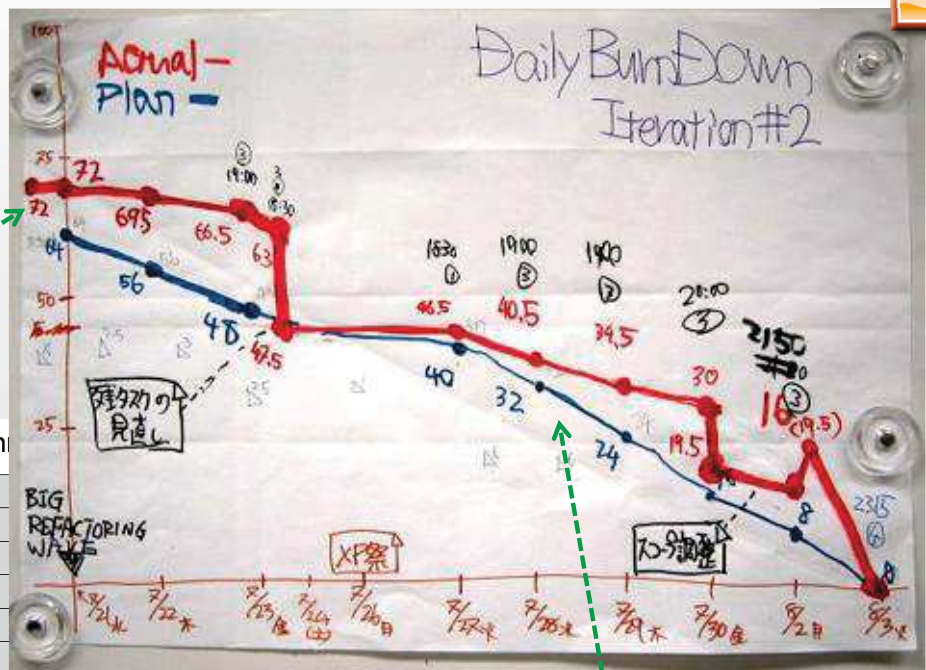
▶ Backlog jako plán



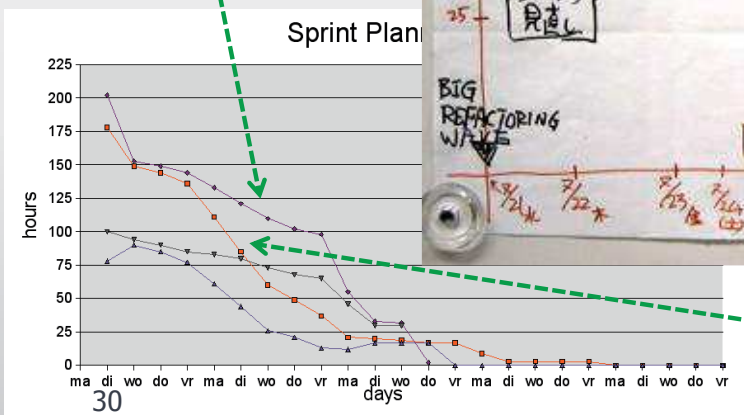
- ▶ Product backlog
 - ▶ epics + user stories = požadavky na produkt
 - ▶ priority, rozpracovanost => pořadí implementace
- ▶ Iterační backlog
 - ▶ stories + tasks = plán iterace
- ▶ Aktivity související s plánováním
 - ▶ **Product** backlog: backlog **grooming**, dot voting
 - ▶ **Iterační** backlog: planning meeting, **daily** standup

- Zahrnuje
- požadavky
 - priority
 - odhadování
 - plánování
 - průběh

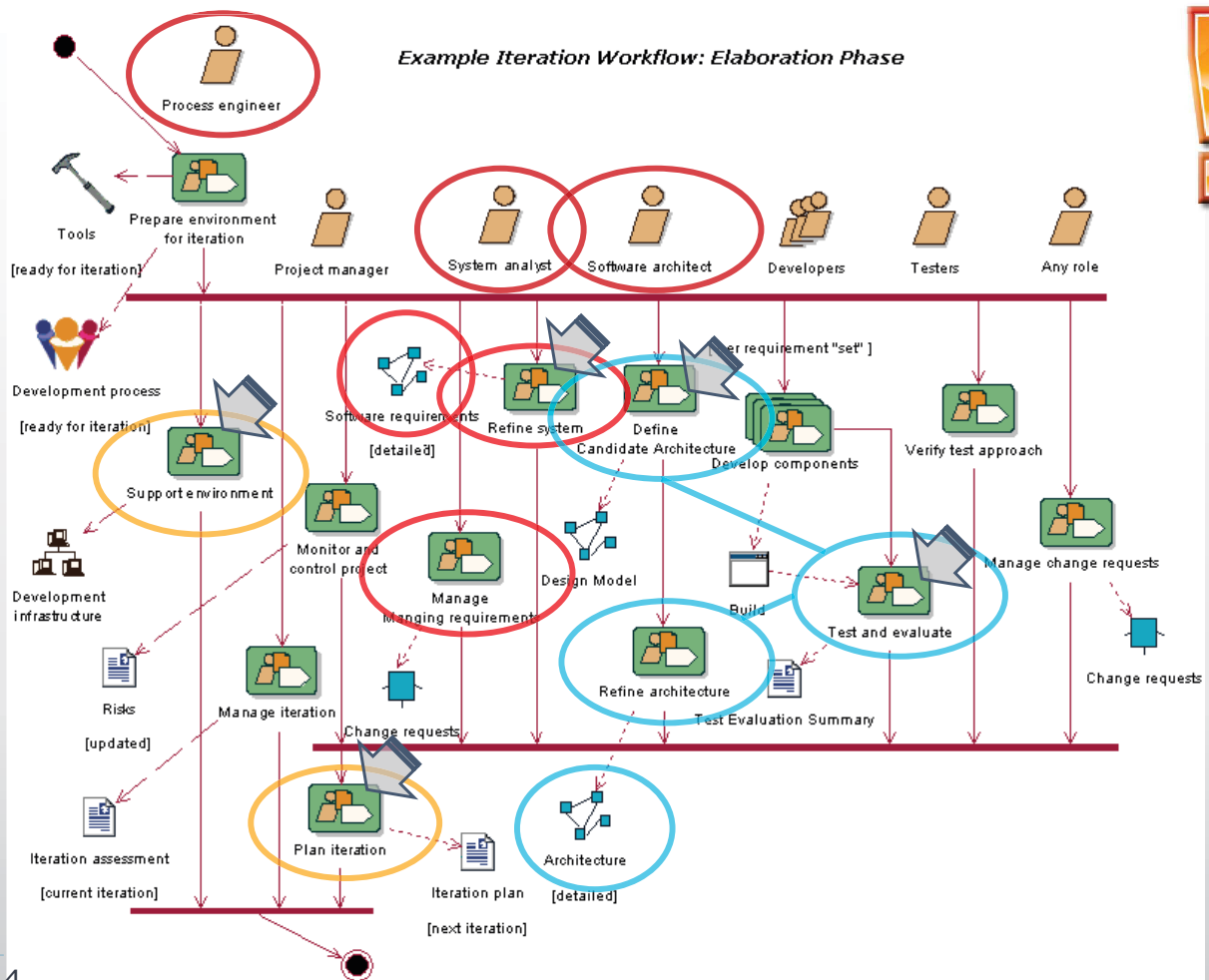
▶ Burndown chart



Kolik práce zbývá



Kolik času je k dispozici



► Milník = kdy jsou cíle fáze dosaženy

► LCA

- Vize a klíčové požadavky jsou stabilní
- testy ověřily, že architektura řeší rizikové požadavky/faktory
- 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 (aktualizace), Specifikace požadavků
- Seznam rizik a strategie jejich řešení (aktualizace)
- Popis architektury, validační testy
- Plán projektu, Popis infrastruktury

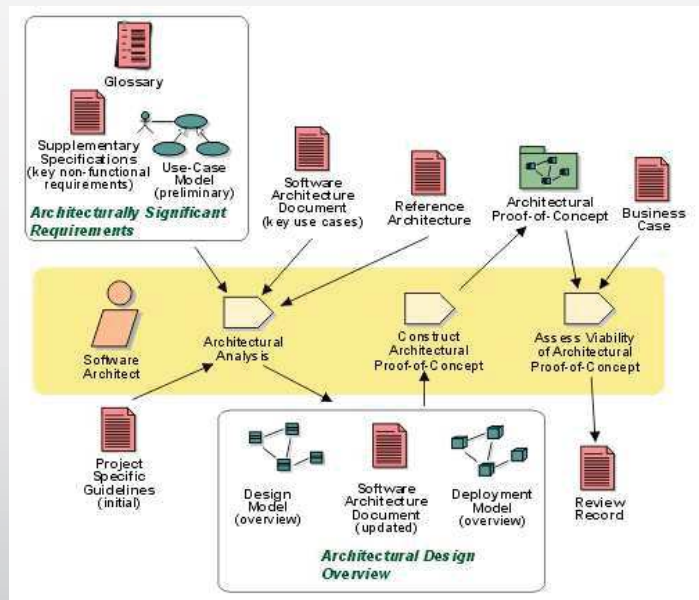
Validace architektury

- „Bude IOC, REL na tomto postavený splňovat LCO?“
 - executable architecture a její validace
 - architektonicky důležitá funkčnost (use cases)

Mechanismy

- návrh na základě klíčových use cases a mimofčních pož.
- referenční architektura
- proof of concept implementace
- oponentura

Samostudium: UP Concept: Executable Architecture



Podrobný popis případu užití

Detailní rozbor komunikace aktér-systém

- Standardní průběh**
 - nejčastější sled akcí
 - bez chyb a různých možností
- Vstupní a výstupní podmínky**
 - co potřebujeme pro standardní průběh
- Chybové stavy a alternativní průběhy**
 - určení míst výskytu, příčin, následků
 - popis alternativních a chybových akcí

```

Název a popis:
PU002 Půjčit exempláře
Umožňuje vlastníkovvi zaevidovat vypůjčení exemplářů

Standardní průběh:
# vlastník zvolí volbu "výpůjčka" v nabídce
# čtenář oznámí vlastníkovvi svoji identifikaci (jmé
# vlastník zadá nebo vyhledá čtenáře v seznamu zamé
<alt: čtenář nenalezen v evidenci>
# systém zobrazí všechny volné exempláře vlastníka
# pro všechny půjčované exempláře
## vlastník vyhledá vypůjčovaný exemplář ve svém fo
podle PU004 Procházet katalog -- omezeno na fond
## systém ověří, že vybraný exemplář je k dispozici
rezervovaný)
<alt: na exemplář je rezervace>
## systém zobrazí návrh výpůjčky s datem vrácení
## vlastník může data návrhu opravit, poté návrh od
## systém vytvoří záznam o vypůjčce exempláře čtená
jeho data podle hodnot upravených vlastníkem
## systém informuje vlasťtka o vytvoření výpůjčky
## vlastník předá exemplář čtenáři
# tento PU končí volbou "ukončit půjčování" zvoleno

Alternativní průběhy:
čtenář nenalezen v evidenci (krok 3)
- systém upozorní vhodným hlášením, tento PU končí
na exemplář je rezervace (krok 5)
- systém to oznámí vhodným hlášením
- tento PU pokračuje krokem 4 - další exemplář k pu

Vstupní podmínky:
(žádné)

Výstupní podmínky:
- exemplář je zapůjčen čtenáři
- je zaevidována výpůjčka
- pro exemplář je nastaven příznak "vypůjčen"
- systém je připraven pro libovolnou další operaci
    
```

Samostudium: UP Artifact: Use Case

► User Stories: obsah



► Popis jedné funkčnosti z pohledu uživatele

- business value
- terminologie

► Hlavní vlastnosti

- stručnost
- ověřovací kritéria



As a student I want to purchase a parking pass so that I can drive to school

Tests:

- undergrad student: 1-term pass for \$100
- grad student: 1-term pass for \$150
- phd: 1-year pass for \$200
- cash payment
- card payment: Visa, MasterCard only
- receipt indicates type, duration, amount paid

► Re prezentace vlastností



► Účel: možnost ověřit splnění v implementaci

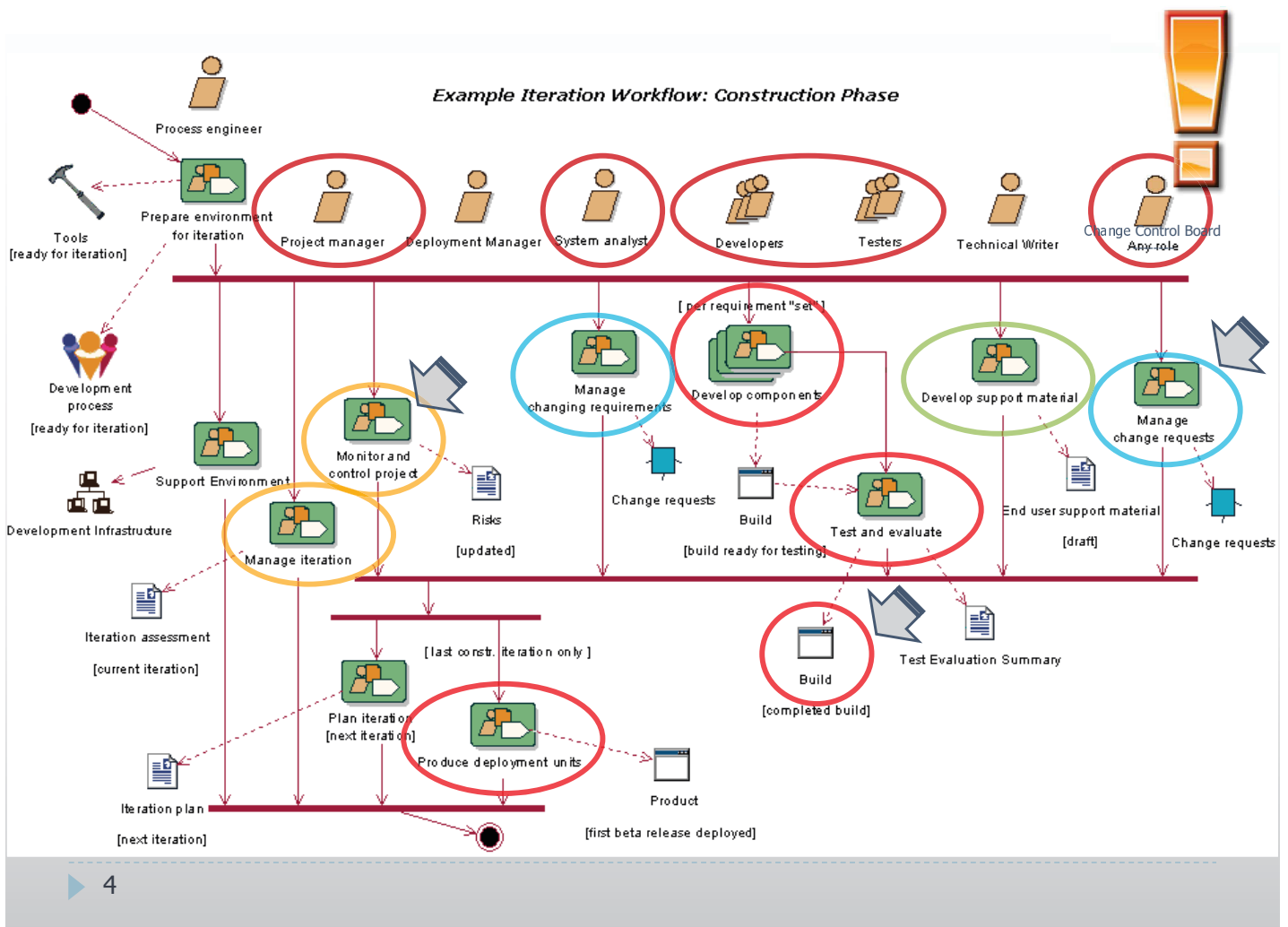
► Měřitelný způsob (numericky)

- obvyklá hodnota, povolené odchylky / četnost / přírůstky
- zvažování realizovatelnosti funkčních požadavků
- vhodná reprezentace pro neměřitelné

► Popis vlastností

- u případů užití
- u tříd problémové oblasti
- vázané na celý systém





▶ Milník = kdy jsou cíle fáze dosaženy

▶ IOC

- ▶ Je hotová „beta“ verze produktu
- ▶ viz **Boehm: Anchoring the Software Process**

▶ Artefakty

- ▶ Architektura (aktualizovaná), popisy implementace
- ▶ Testovací sady + reporty
- ▶ Plán nasazení (první verze)
- ▶ Uživatelská příručka a podpůrné materiály (draft)

► Cyklus správy změn

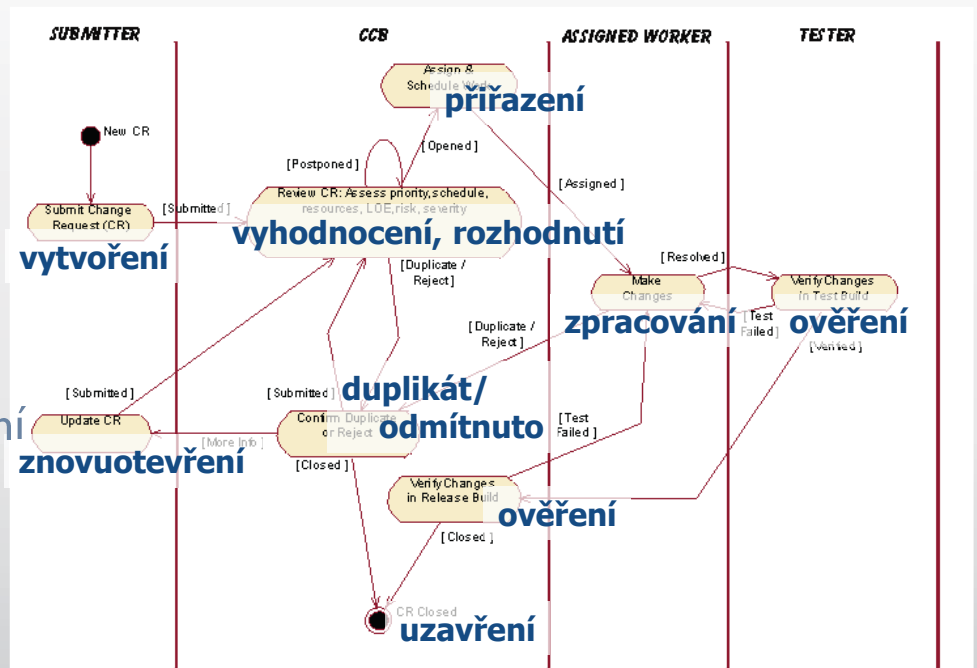


► Požadavek na změnu – stavy

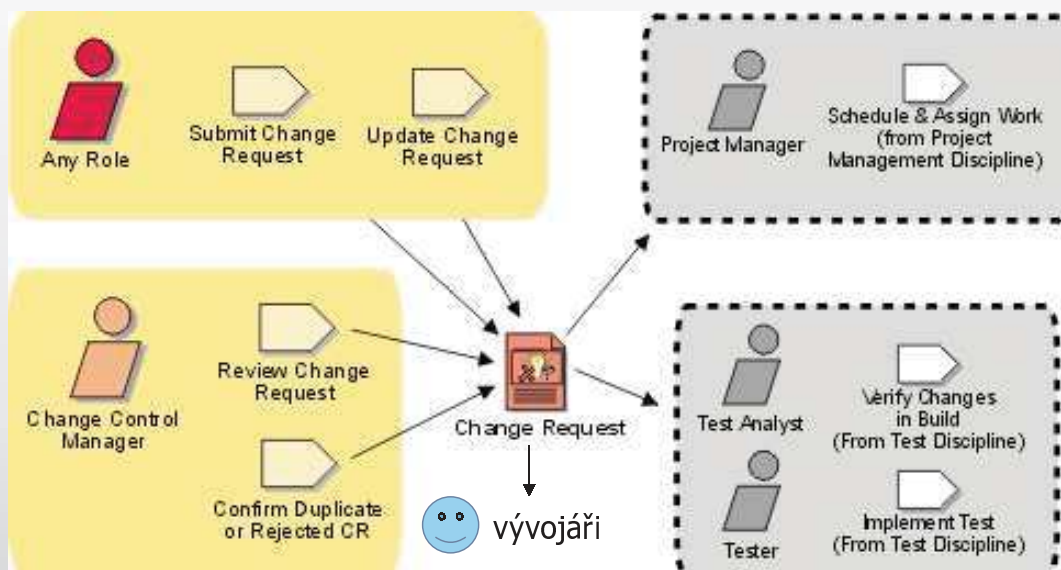
- vytvořený
- schválený
- přiřazený
- vyřešený
- ověřený
- uzavřený
- znovuotevřený
- odmítnutý

► Během provádění

- znovuotevření problémů
- vygenerování nových hlášení



► Role ve správě změn



► Určení konkrétní verze prvku



► Základní druhy verzí

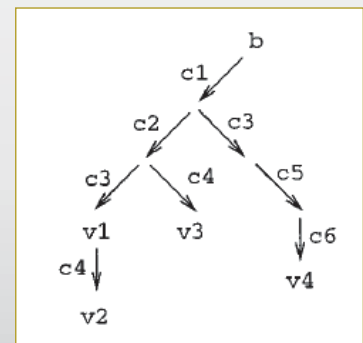
- historická podoba → **revize** („Word 6.0“)
- alternativní podoba → **varianta** („Word pro Macintosh“)

► Verzování podle stavu

- identifikují se pouze podoby prvků
- výsledná verze vznikne vhodným výběrem

► Verzování podle změn

- identifikují se (také) změny prvků
= **changeset**
- výsledná verze vznikne aplikací změn



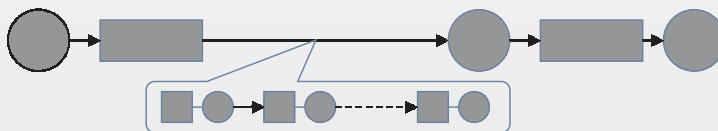
32

► Terminologie: Codeline



► Codeline (vývojová linie) = série podob (verzí) množiny prvků konfigurace tak, jak se mění v čase

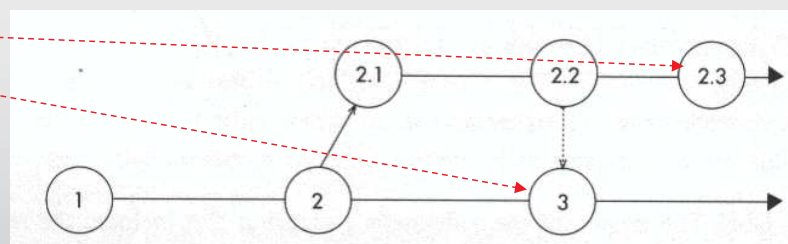
- (extenzionální verzování podle stavu) obsahuje **commity** a/nebo **changesety**, větve => část grafu verzí



- vrchol codeline obsahuje nejčerstvější verzi

□ “HEAD”, “tip”

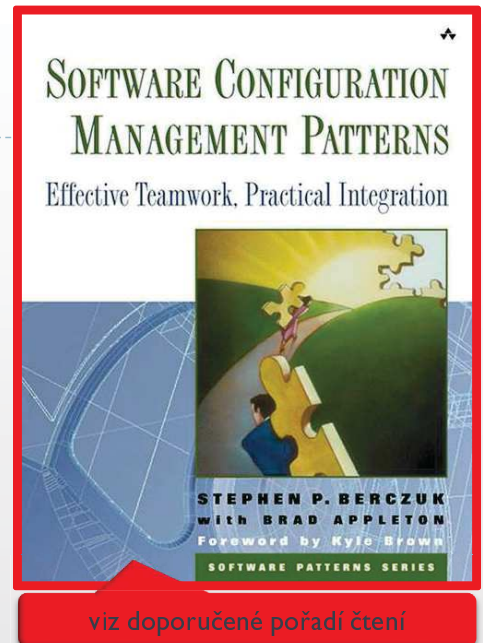
- má přiřazena **pravidla** práce na codeline



41

▶ Vzory pro verzování (1)

- ▶ Hlavní vývojová linie (*mainline*) a pravidla linie (*codeline policy*)
 - ▶ jedna codeline pro průběžný vývoj
- ▶ Stabilizační období (*code freeze*)
 - ▶ pravidla před release
- ▶ Větev pro release a jeho přípravu (*release line*)
 - ▶ místo code freeze mít samostatnou větev pro release
- ▶ Kód třetích stran na větev (*third party codeline*)
 - ▶ vlastní větev pro každý kód od externího dodavatele



51

▶ Vzory pro verzování (2)

- ▶ Privátní verze (*private versions*)
 - ▶ soukromé úložiště pro častější check-in
- ▶ Větev pro úkol (*task branch*)
 - ▶ složitější úpravy s většími následky dělat na větvi
 - ▶ vazba na správu změn
- ▶ Check-in pro každý úkol (*task-level commit*)
 - ▶ minimum nutného
 - ▶ po ukončení práce na jednom úkolu udělat check-in změn

A successful Git branching model
<http://nvie.com/posts/a-successful-git-branching-model/>



52

▶ Vlastnosti sestavení



▶ Jedinečnost a identifikovatelnost

□ PROJEKT_v2_build2134_20041220T1954

- ▶ identifikátor jednoznačný, čitelný
- ▶ vytvořitelný a zpracovatelný automaticky (schema pro id)

▶ Úplnost

- ▶ tvoří kompletní systém, obsahuje všechny komponenty

▶ Konzistence

- ▶ vzniklo ze správných verzí správných komponent

□ tj. z konzistentní konfigurace

▶ Opakovatelnost

- ▶ možnost opakovat build daného sestavení kdykoli v budoucnu

□ se stejným výsledkem

▶ Dodržuje pravidla vývojové linie

- ▶ build odpovídající baseline
- ▶ zejména release má striktní pravidla

69

▶ Integrační sestavení

▶ Cíl: spolehlivě ověřit, že produkt jde sestavit

- ▶ soukromý build nestačí
 - složité závislosti, špecifiká ve workspace, zjednodušení pro zrychlení
- ▶ úplné sestavení trvá dlouho => nemůže provádět vývojář

Microsoft Windows NT 3.0 consisted of 5.6 million lines of code spread across 40,000 source files. A complete build took as many as 19 hours on several machines, but the NT development team still managed to build every day.
– McConnell, 1996

▶ Postup: celý produkt (vč. závislostí) sestaven centrálně, automatizovaným a opakovatelným procesem

- ▶ postup co nejpodobnější sestavení pro release
 - vždy „na zelené louce“ (clean full build)
- ▶ maximální automatizace - typicky běží přes noc
- ▶ spolehlivé mechanismy zaznamenání chyb a informování o nich
 - emailové notifikace začátek, konec, výsledek
 - web s přehledy a detaily
- ▶ úspěšné sestavení může být označováno ve verzovacím systému



75

▶ Daily Build and Smoke test



- ▶ Integrovaní sestavení + zkouška těsnosti
 - ▶ pravidelně 1x denně (někdy nočně)
 - ▶ výsledky okamžitě známy a reflektovány
 - nová hlášení problémů
 - opravy ihned zapracovány do kódu
 - ▶ check-in kódu, který vede k chybám, je neslušné chování
 - lehká (nebo i vážnější) sankce vhodná
- ▶ Výhody
 - ▶ malé množství změn během denních check-in

=> zvladatelné množství oprav, **včas detekce problémů „vždyt’ včera to fungovalo“**, **analýza změn kódu místo ladění – viz diskuse o sestavení**

 - ▶ pravidelný, obecně známý rytmus projektu
 - ▶ lepší morálka týmu („to nám to roste“)
- ▶ Cena: trocha disciplíny, trocha automatizace

86

▶ Goal-Question-Metric



- ▶ Přístup k definování metrik
 - Basili et al 1992
 - rámec pro systém zaměřený na konkrétní problémy

Solingen, R. van, Basili, V., Caldiera, G., & Rombach, H. D. **Goal Question Metric (GQM) Approach**. In Encyclopedia of Software Engineering. John Wiley & Sons, 2002

- ▶ Goal – problém + cíl měřicího programu
- ▶ Question – měřené objekty a způsob měření
- ▶ Metric – konkretizují získávaná data
 - ▶ G: Zlepšit spravedlivost v oceňování práce na projektu
 - ▶ Q: Kolik práce odvádí jednotliví členové týmu?
 - ▶ M: Počet řádek uložených v svn; Váha uzavřených tasků v bugtrackeru (součty severity*effort)

▶ Milník = kdy jsou cíle fáze dosaženy

▶ GA (Product Release)

- ▶ „Result of customer reviewing and accepting the deliverables“
- ▶ viz **Boehm: Anchoring the Software Process**

▶ Artefakty

- ▶ Release produktu
- ▶ Podpůrné materiály („uživatelská dokumentace“)
- ▶ Baseline kompletní konfigurace release
- ▶ ... dle povahy produktu

