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

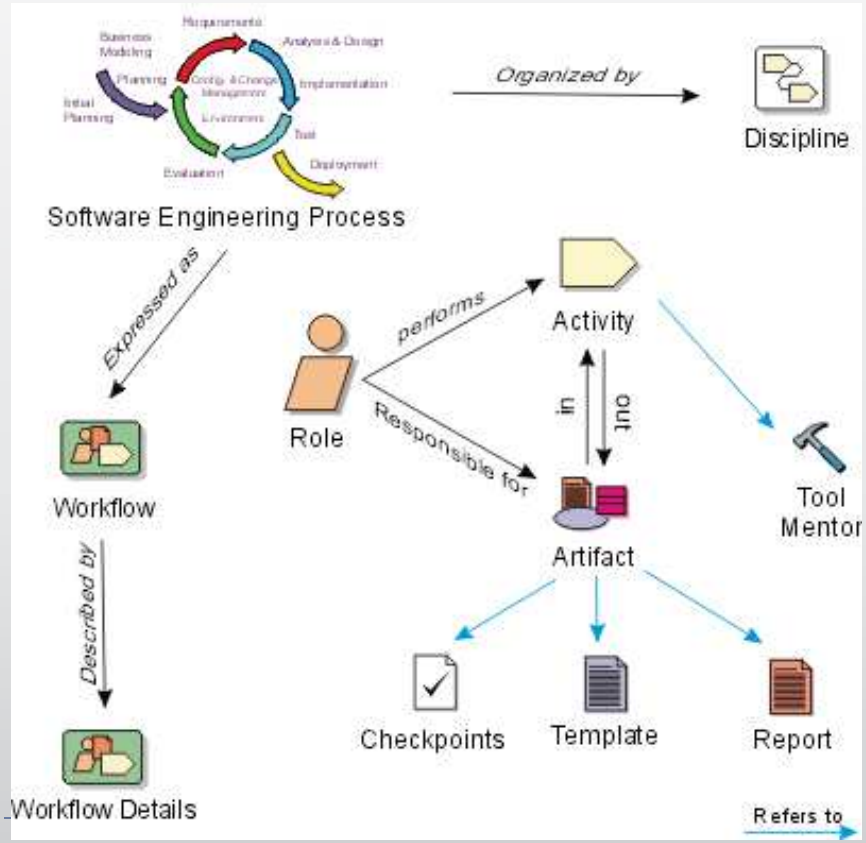
Komerční zákazník
Státní sféra
Vertikály (utility, banky, telco, …)

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



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

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

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

Viz RUP Milestone: Lifecycle Objectives

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

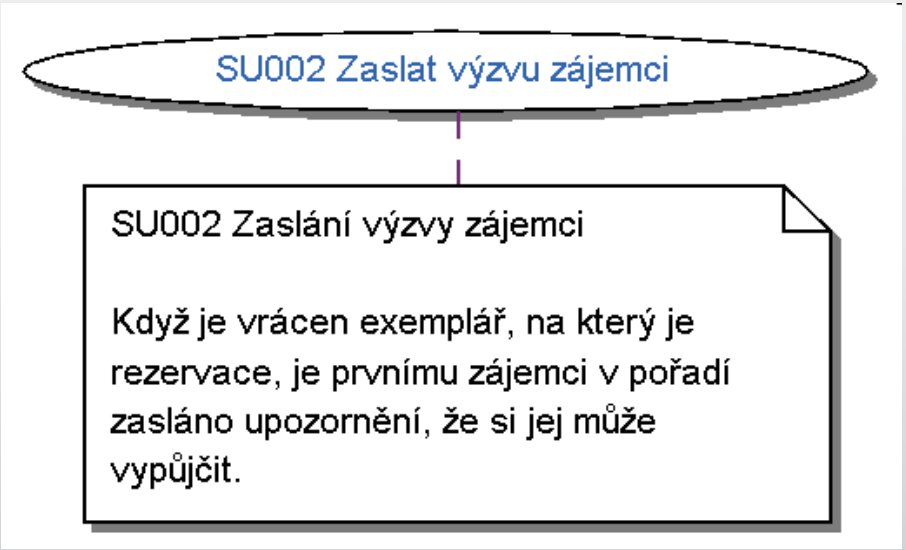
Příklad: Wiegers

Základní popis případu užití

… ve fázi shromažďování požadavků:
základní popis dané funkce aplikace

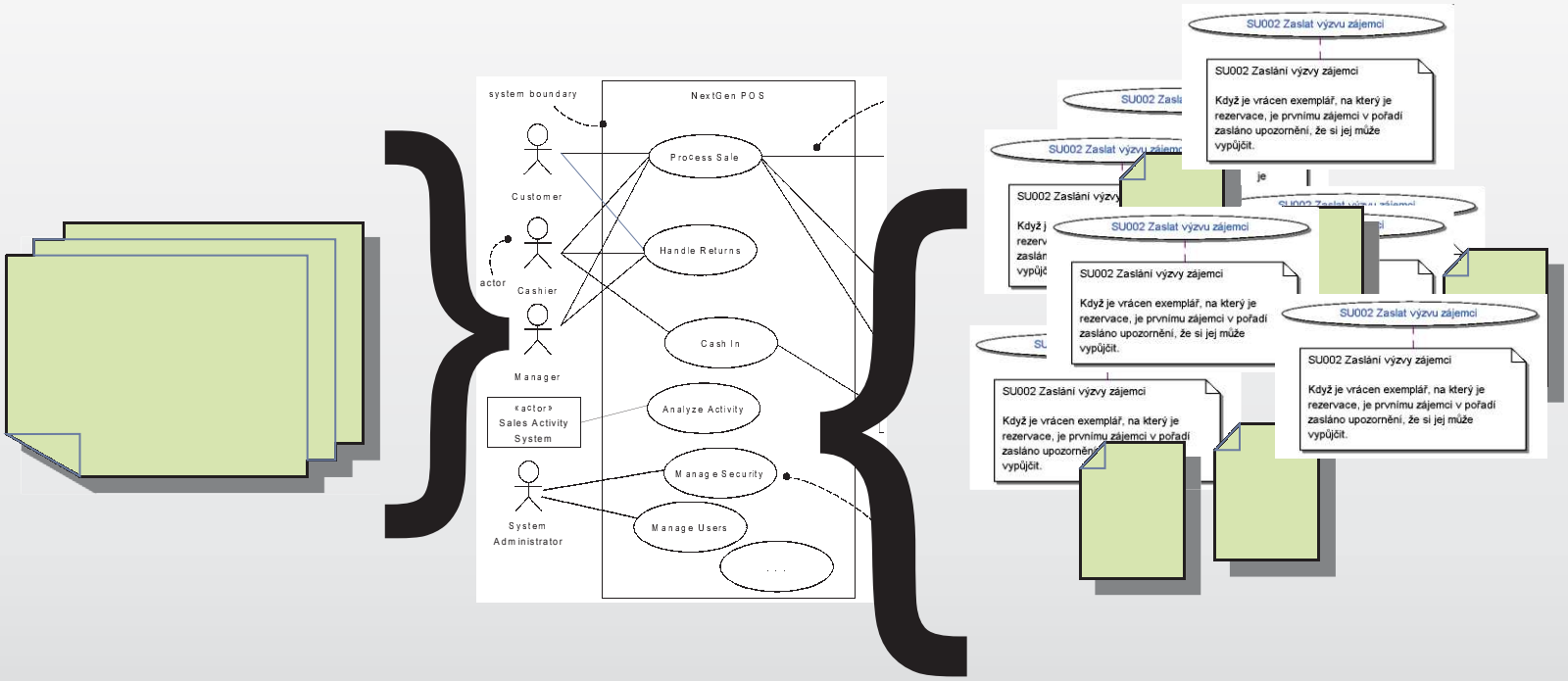
- Název (+ ID)
- Stručný popis – 1 věta

- Případně
  - Základní kroky postupu pro klíčové PU
  - Odkazy na zdrojové informace



Obrysy požadavků s případy užití

- Charakteristiky aktérů
- Model užití



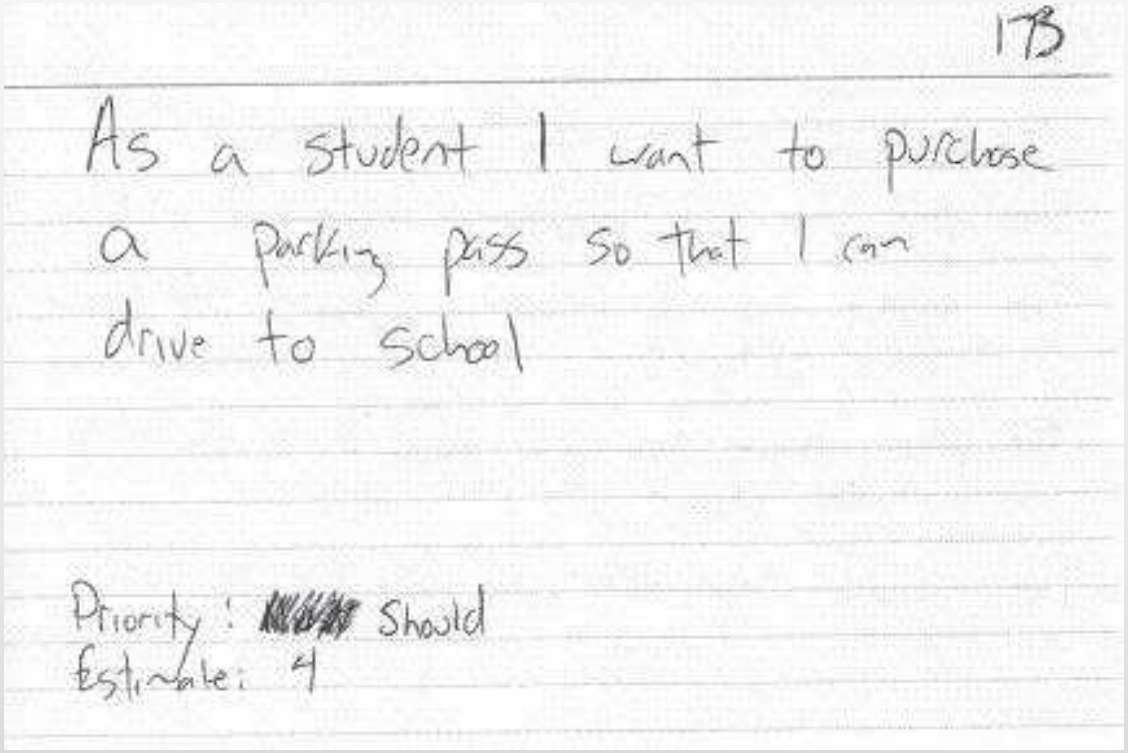
- Jeden dokument (diagram jako „obsah“) nebo informace v UML nástroji

Forma: User Story

- Co uživatel od systému očekává a proč

- Obsah
  - název
  - stručný popis
  - důležitost

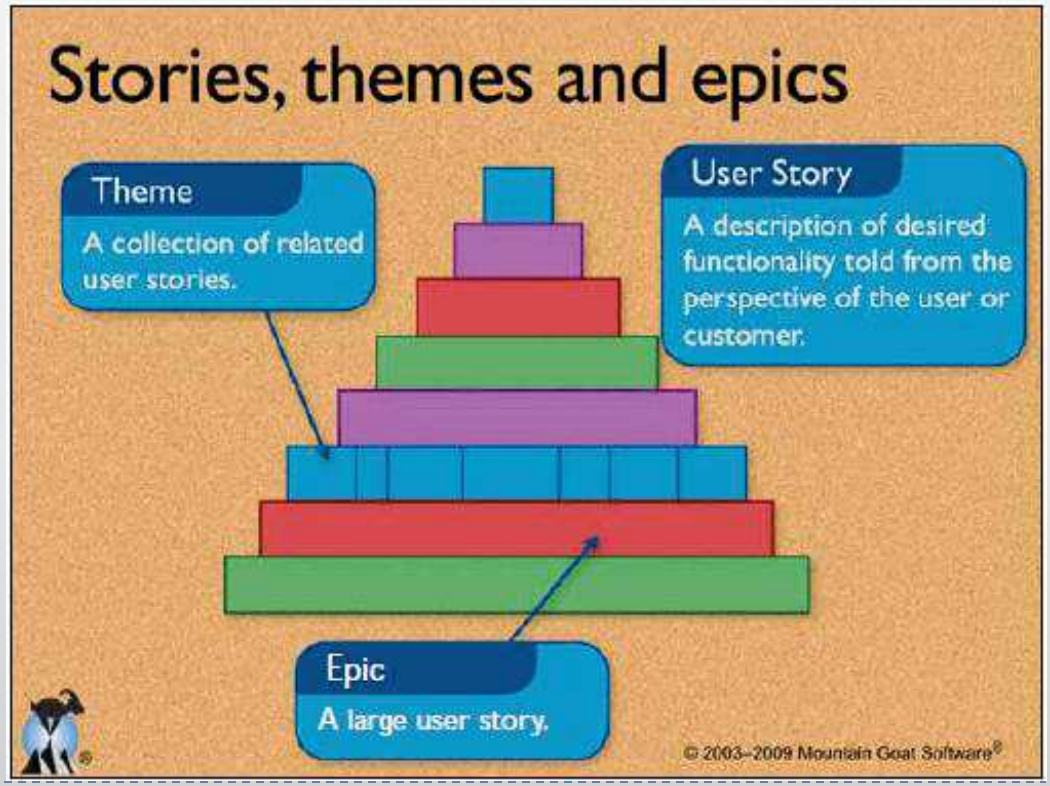
- Způsob zápisu
  - karta
  - položka v ALM nástroji



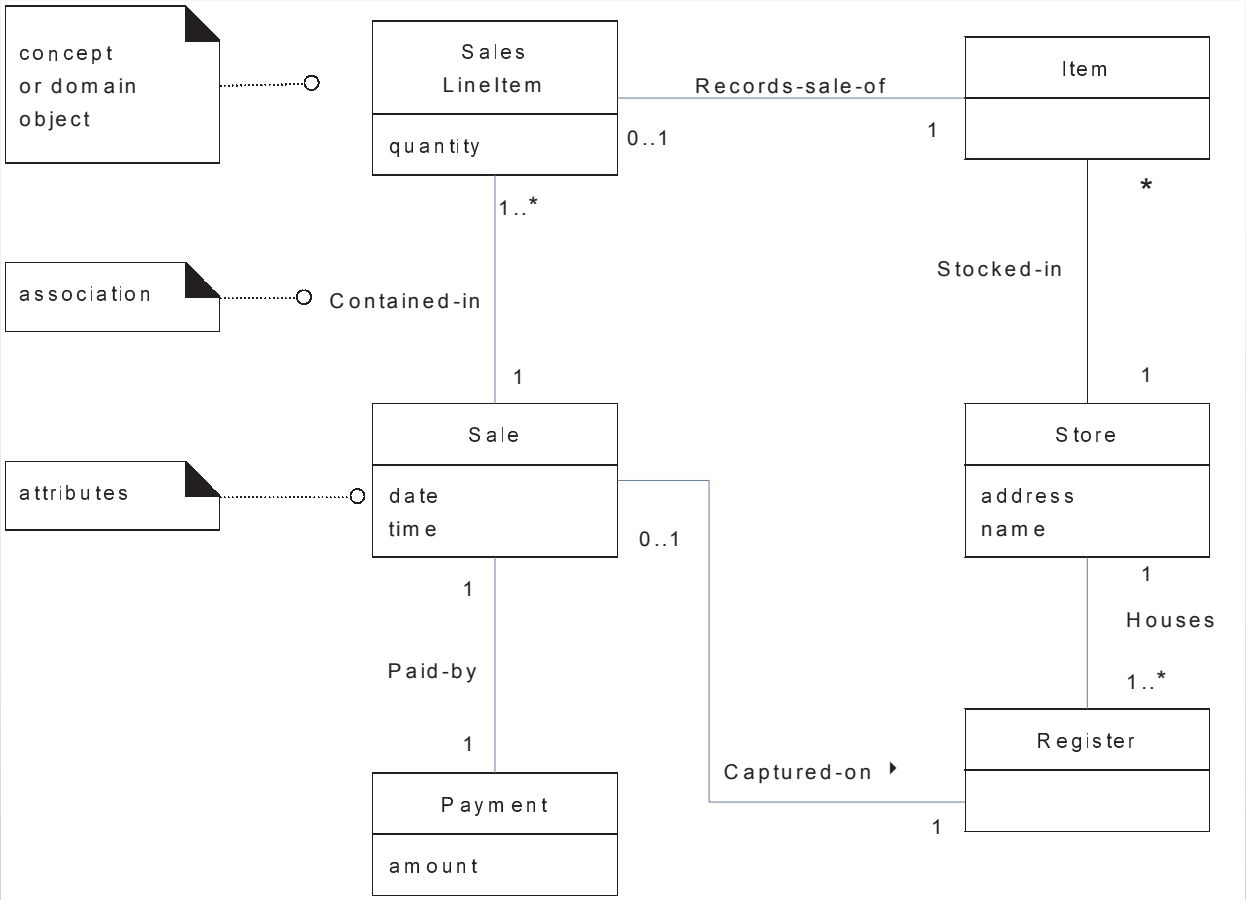
http://www.agileconnection.com/article/how-do-i-write-requirements-using-stories-and-acceptance-criteria-part-one
http://www.agilemodeling.com/artifacts/userStory.htm

Obrysy požadavků agilně: Backlog

- Product Backlog = základní struktura
  - obsahuje epics, stories
  - just-in-time zpřesňování (příští iterace)
- Nejen požadavky
  - viz Plánování

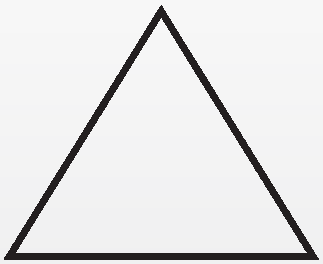


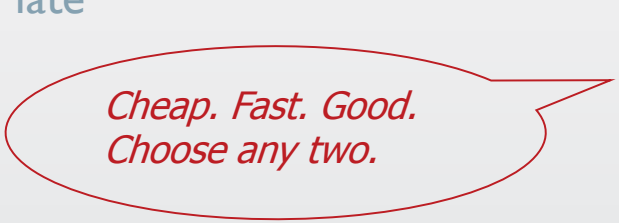
Obrysy základních struktur s UML: doménový diagram



Stupně volnosti při plánování

- Klasicky: čas, zdroje (cena), kvalita
  - obtížně měnitelné, odhadované
  - kvalita obtížně řiditelná
    - 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

http://radiantminds.com/basic-laws-of-planning/

Backlog jako plán

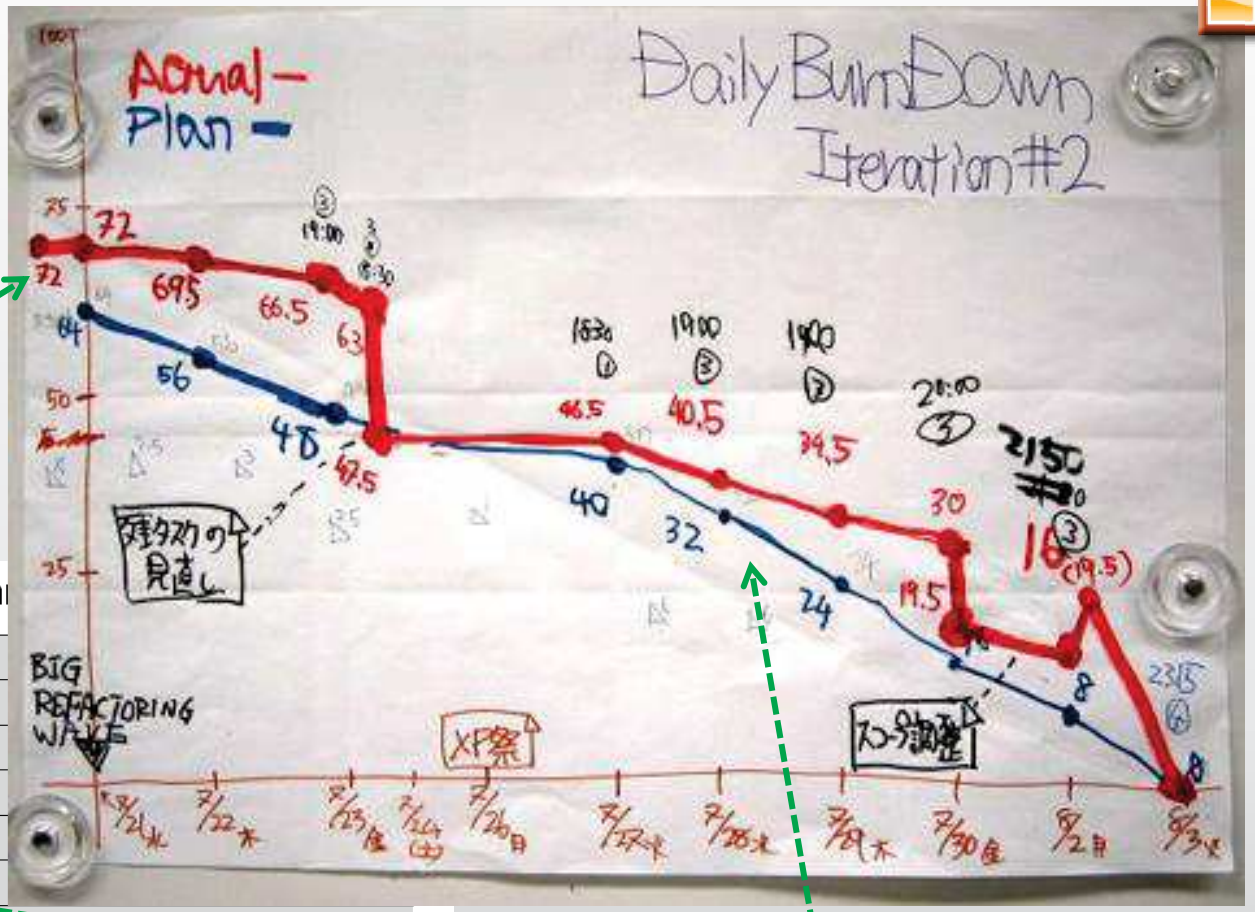
Zahrnuje
- požadavky
- priority
- odhadování
- plánování
- průběh

- Product backlog
  - epics + user stories = požadavky na produkt
  - priority, rozpracovanost => pořadí implementace
- Iterační backlog
  - stories + tasks = plán iterace

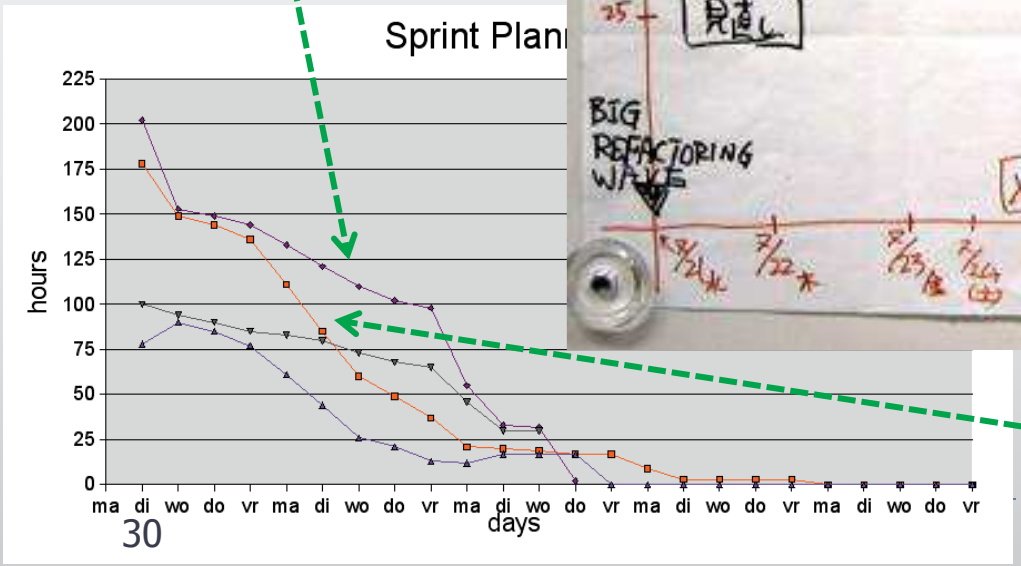
- Aktivity související s plánováním
  - Product backlog: backlog **grooming**, dot voting
  - Iterační backlog: planning meeting, **daily** standup

http://www.agileconnection.com/article/grooming-product-backlog

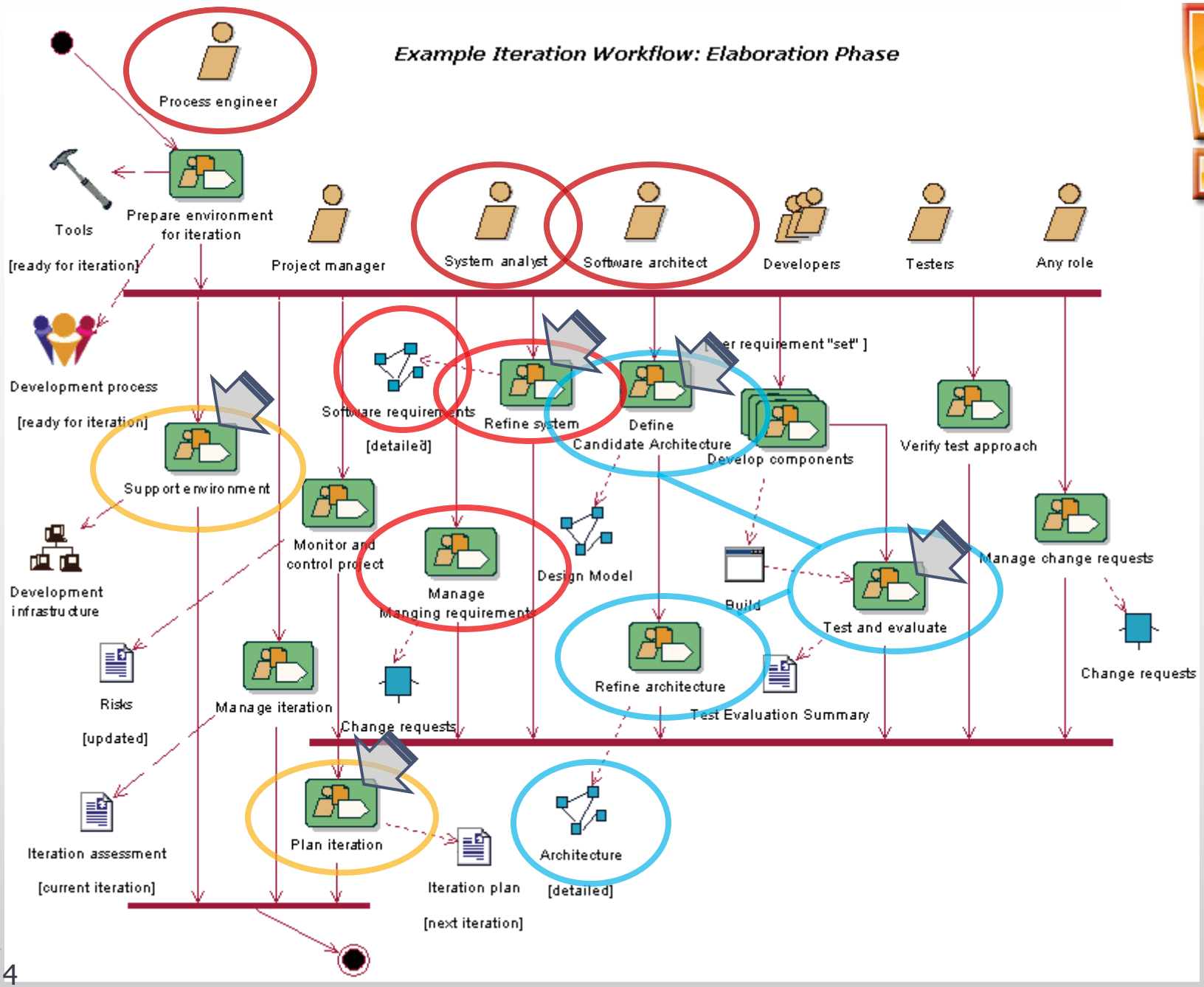
Burndown chart



Kolik práce zbývá



Kolik času je k dispozici


Example Iteration Workflow: Elaboration Phase
Process engineer
Tools
Prepare environment for iteration
[ready for iteration]
Project manager
System analyst
Software architect
Developers
Testers
Any role
Development process
[ready for iteration]
Software requirements
[detailed]
Refine system
Define Candidate Architecture
Develop components
Verify test approach
Support environment
Monitor and control project
Manage changing requirements
Design Model
Build
Test and evaluate
Manage change requests
Development infrastructure
Risks
[updated]
Manage iteration
Change requests
Refine architecture
Test Evaluation Summary
Change requests
Iteration assessment
[current iteration]
Plan iteration
Iteration plan
[next iteration]
Architecture
[detailed]

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

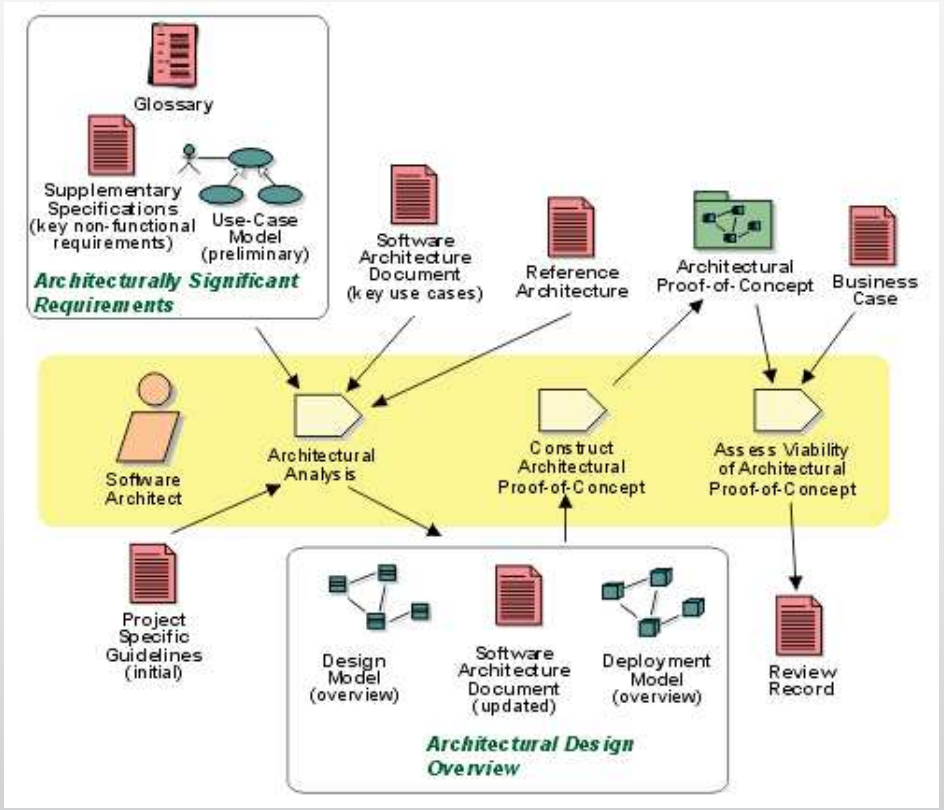
Viz RUP Milestone: Lifecycle Architecture

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: Exectuable Architecture



Podrobný popis případu užití

Detailní rozbor komunikace aktér-systém

① Standardní průběh
- nejčastější sled akcí
- bez chyb a různých možností

② Vstupní a výstupní podmínky
- co potřebujeme pro standardní průběh

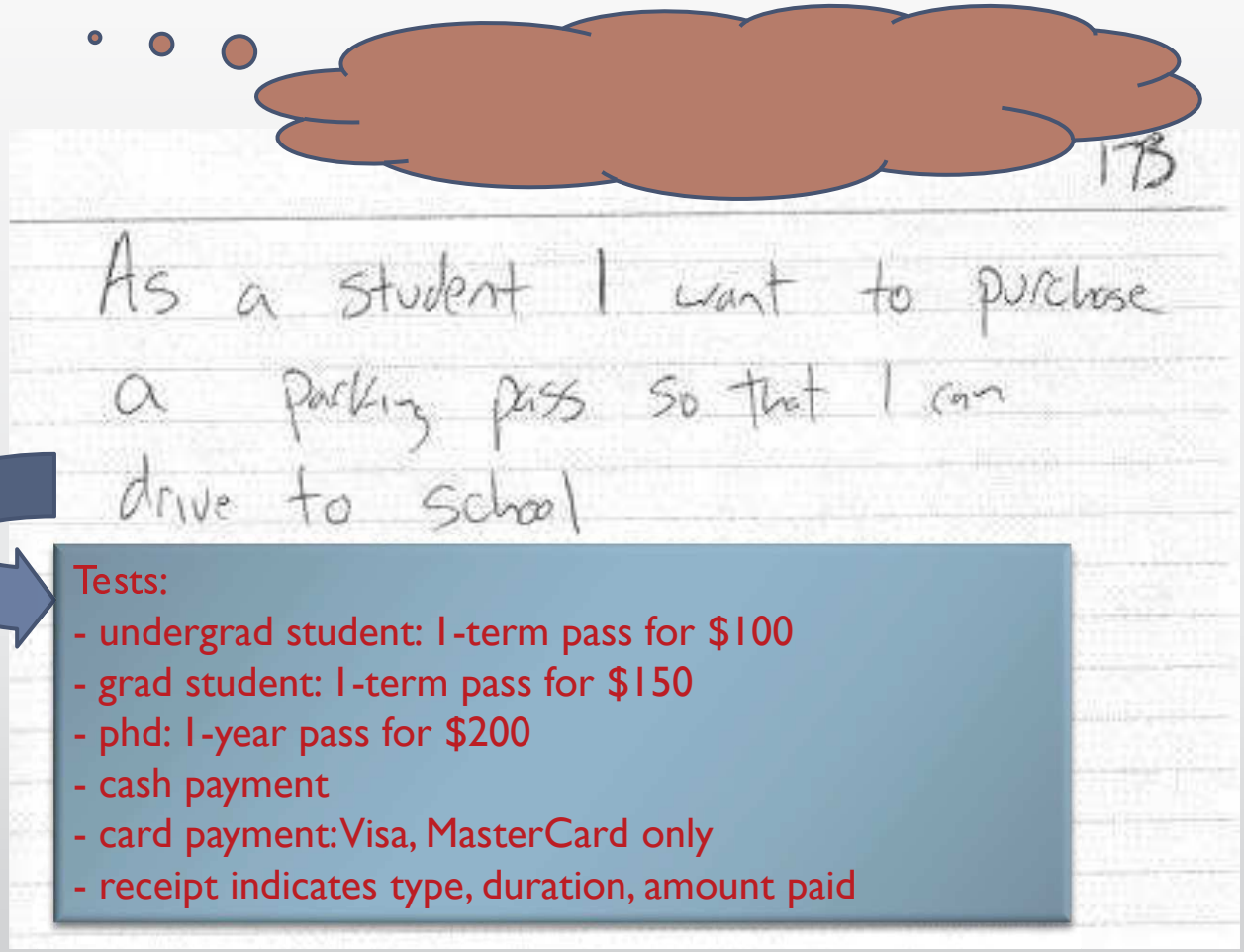
③ Chybové stavy a alternativní průběhy
- určení míst výskytu, příčin, následků
- popis alternativních a chybových akcí

```
Název a popis:

 PU002 Půjčit exempláře

 Umožňuje vlastníkovi zaevidovat vypůjčení exemplářů


Standardní průběh:

 # vlastník zvolí volbu "výpůjčka" v nabídce
 # čtenář oznámí vlastníkovi svoji identifikaci (jmé
 # vlastník zadá nebo vyhledá čtenáře v seznamu zamě
 <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 datumem vrácení
 ## vlastník může data návrhu opravit, poté návrh od
 ## systém vytvoří záznam o výpůjčce exempláře čtená
    jeho data podle hodnot upravených vlastníkem
 ## systém informuje vlasntíka 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 pů


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



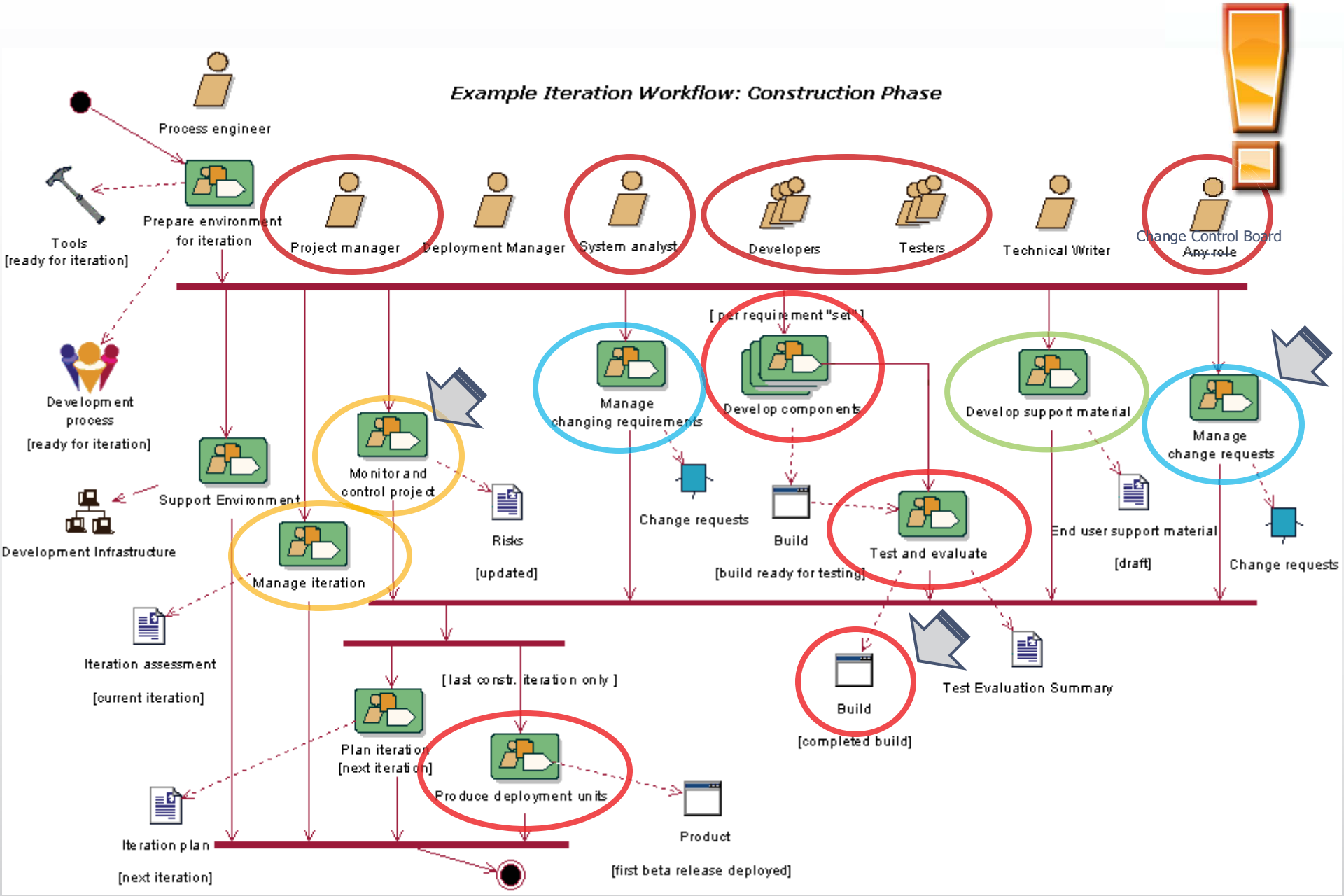
http://www.agilemodeling.com/artifacts/userStory.htm
http://www.agileconnection.com/article/how-do-i-write-requirements-using-stories-and-acceptance-criteria-part-two

Reprezentace vlastností

- Účel: možnost ověřit splnění v implementaci

- Měřitelný způsob (numericky)
  - obvyklá hodnota, povolené odchylky / četnost / přírůstky
  - zvážení realizovatelosti funkčních požadavků
  - vhodná reprezentace pro neměřitelné
- Popis vlastností
  - u případů užití
  - u tříd problémové oblasti
  - vázané na celý systém

Doplňkové info: Wiegers „COS SRS document"


Example Iteration Workflow: Construction Phase
Process engineer
Tools
[ready for iteration]
Prepare environment for iteration
Project manager
Deployment Manager
System analyst
Developers
Testers
Technical Writer
Change Control Board
Any role
Development process
[ready for iteration]
Support Environment
Development Infrastructure
Monitor and control project
Risks
[updated]
Manage iteration
Iteration assessment
[current iteration]
Manage changing requirements
Change requests
[per requirement "set"]
Develop components
Build
[build ready for testing]
Test and evaluate
Develop support material
End user support material
[draft]
Manage change requests
Change requests
[last constr. iteration only]
Plan iteration
[next iteration]
Produce deployment units
Iteration plan
[next iteration]
Product
[first beta release deployed]
Build
[completed build]
Test Evaluation Summary

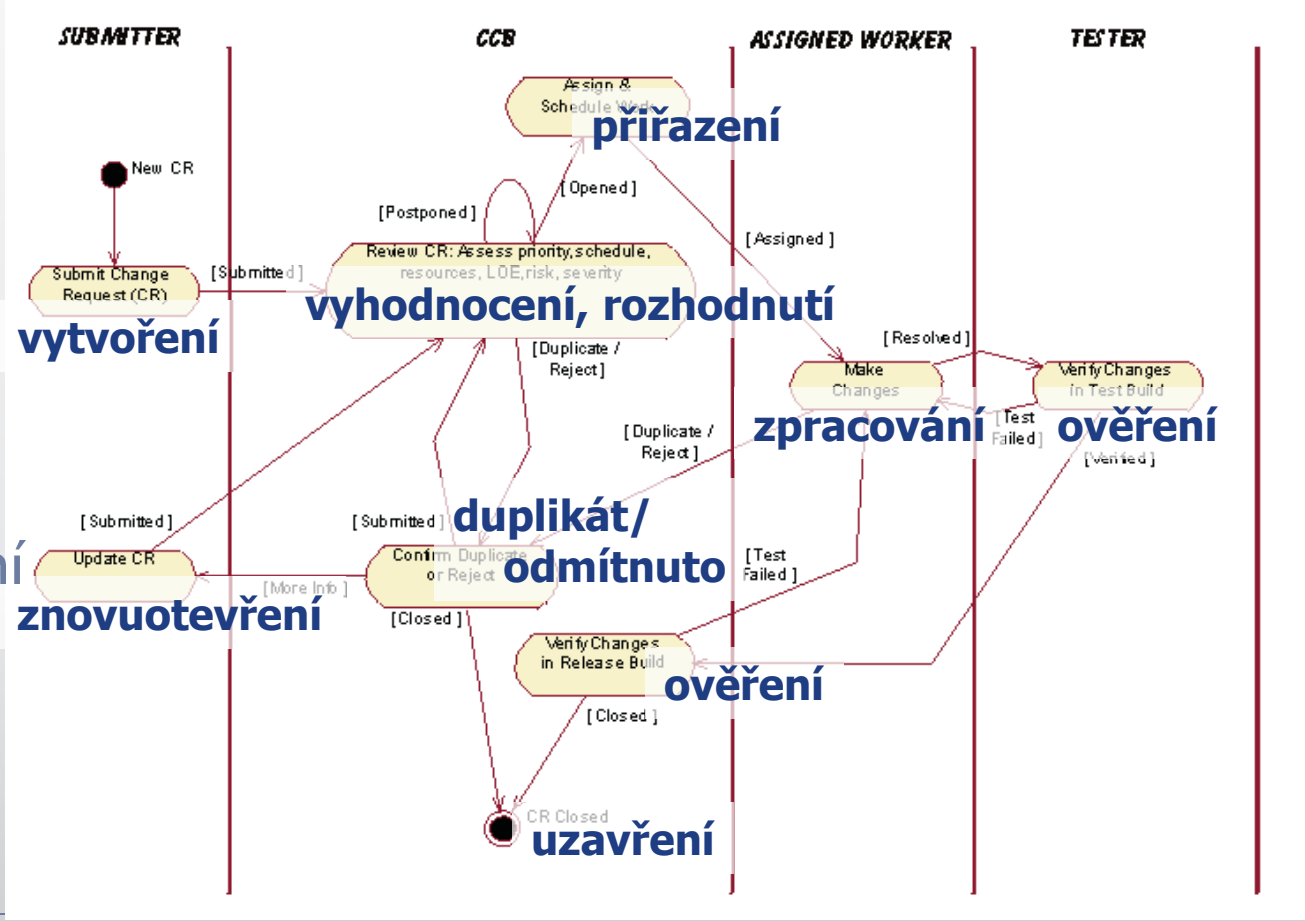
Milník = kdy jsou cíle fáze dosaženy

- 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)

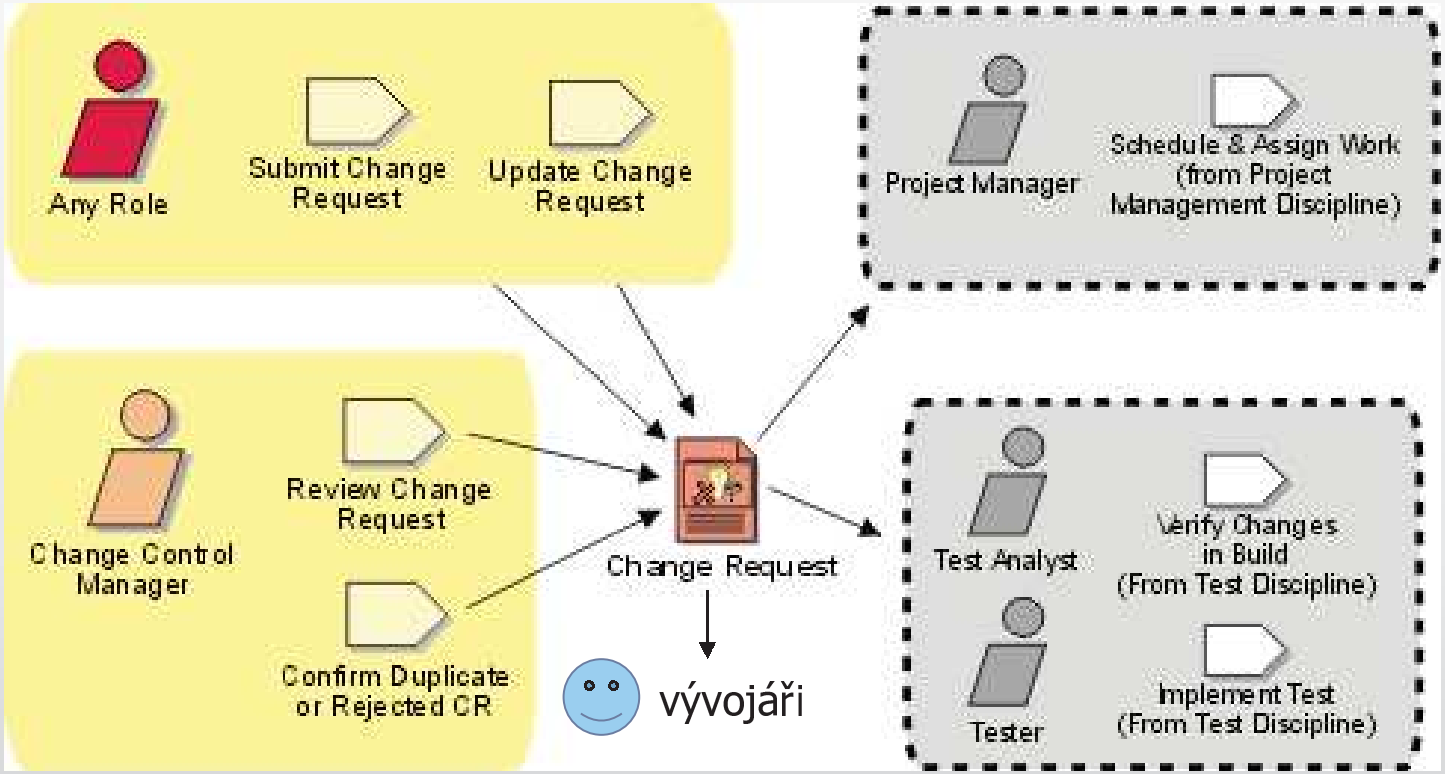
Cyklus správy změn

- Požadavek na změnu – stavy
  - vytvořený
  - schválený
  - přiřazený
  - vyřešený
  - ověřený
  - uzavřený
  - znovuotevřený
  - odmítnutý

- Během provádění
  - znovuotevření problémů
  - vygenerování nových hlášení

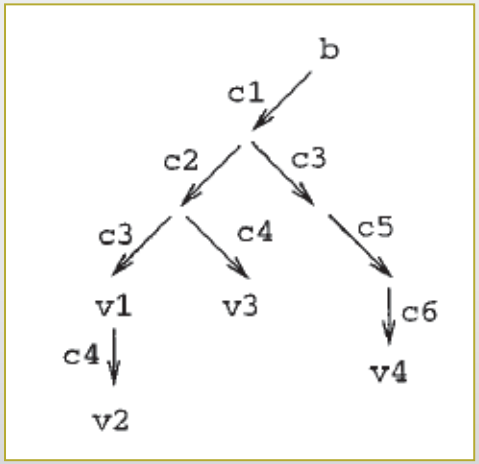


Role ve správě změn



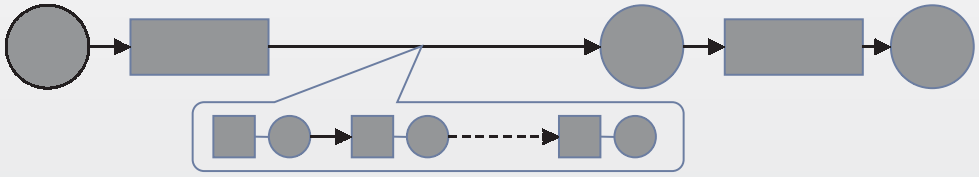
Určení konkrétní verze prvku

- 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

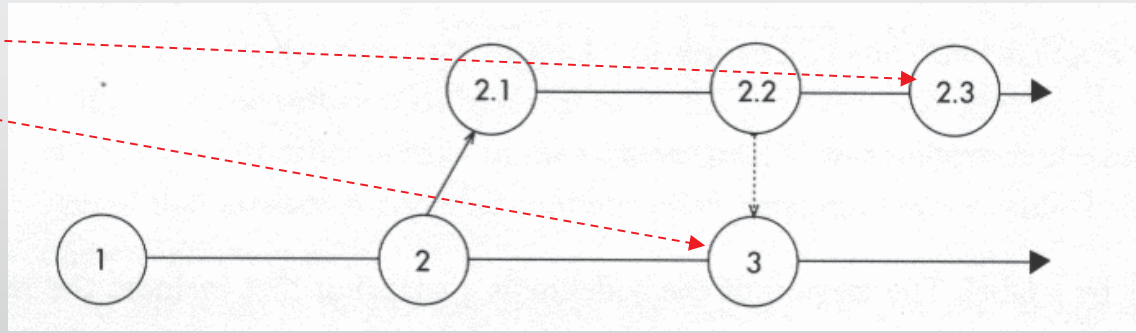


Terminologie: Codeline

- 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

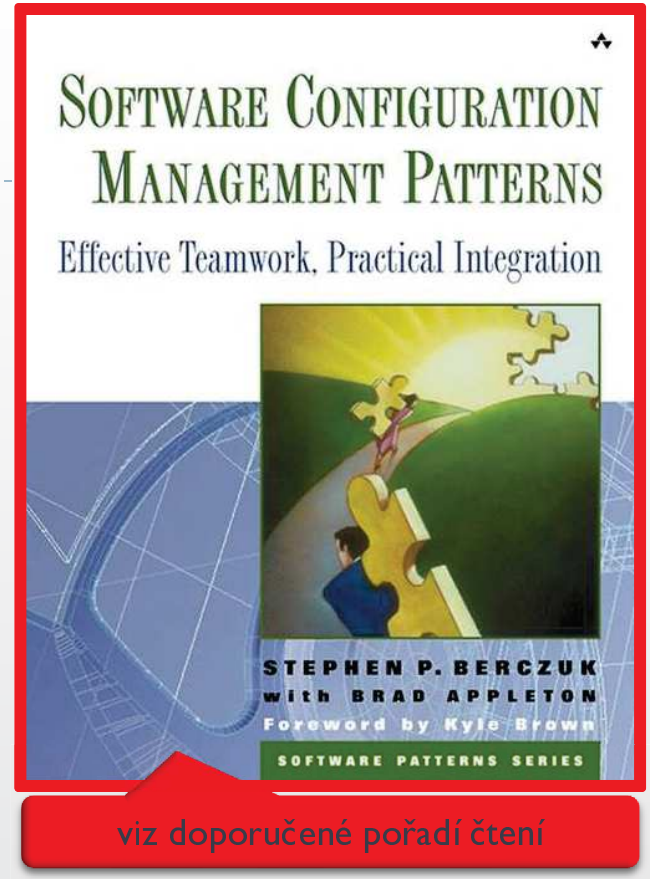


  - vrchol codeline obsahuje nejčerstvější verzi
    - "HEAD", "tip"
  - má přiřazena pravidla práce na codeline



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



viz doporučené pořadí čtení

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/

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

Integrační sestavení

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

- 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

- 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 označkováno ve verzovacím systému

Daily Build and Smoke test

- Integrační 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 „vždyť 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 discipliny, trocha automatizace

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