
Strategie vývoje softwaru

**objektové modelování, iterativní/inkrementální vývoj,
agilní/rigorózní metodiky, RUP**

© Radek Ošlejšek
Fakulta informatiky MU
oslejsek@fi.muni.cz

Organizační záležitosti

Předpokládané znalosti:

- Základy objektového programování
 - např. kurz základní Javy, C++, C#
- Základy modelování informačních systémů a UML
 - PB007 – AnANaS

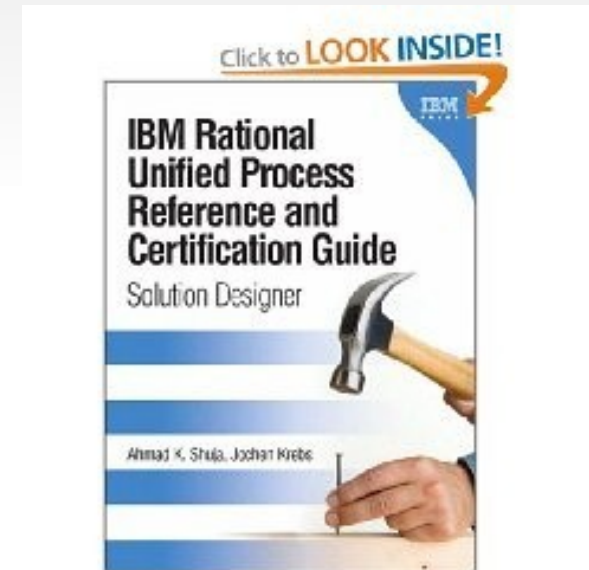
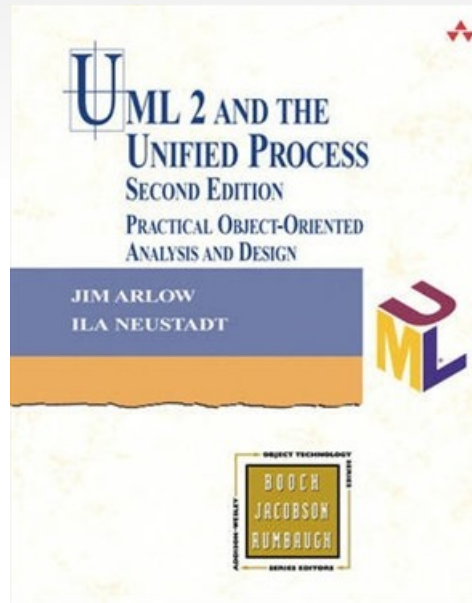
Vhodné znalosti:

- PA104 – Vedení týmového projektu
- PV165 – Procesní řízení
- **PV167 – Projekt z objektového návrhu informačních systémů**

Organizační záležitosti – zkouška

Doporučená literatura:

- Jim Arlow, Ila: UML 2.0 and the unified process – practical object-oriented analysis and design. 2nd ed. Boston : Addison-Wesley, 2005.
- Ahmad K. Shuja, Jochen Krebs: IBM Rational Unified Process Reference and Certification Guide
- ...



Zkouška:

- Písemná, 90 minut. Tři otázky teoretické, jedna praktická.
- Hodnocení: A: 40-34 B: 33-29 C: 28–24 D: 23-20 E: 19-16 F: 15-0

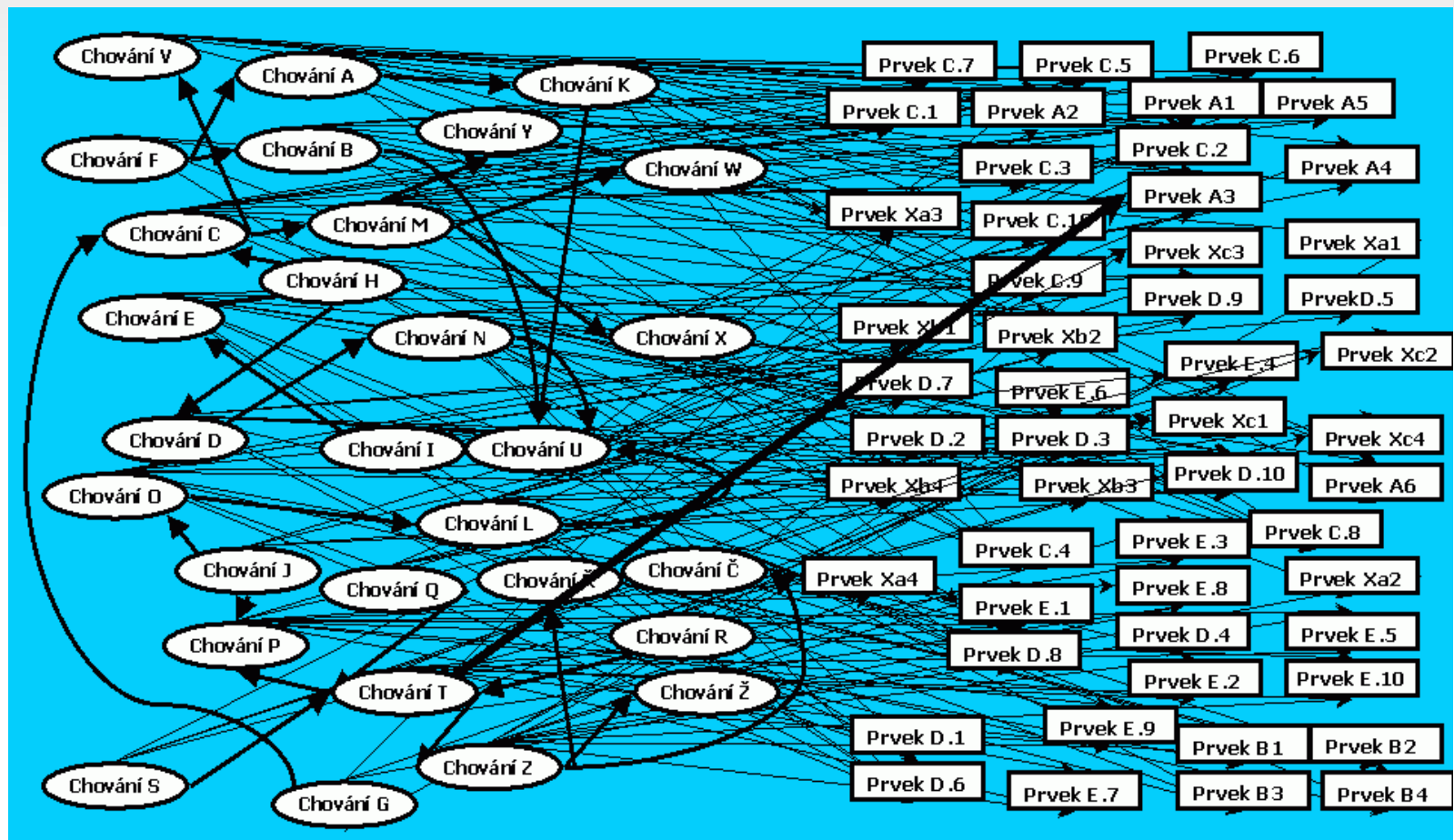
Různé pohledy na strategii vývoje SW

- Proč vůbec modelovat? Modelovat strukturovaně nebo objektově?
- Jaký použít model životního cyklu, tj. návaznost činností při modelování? Vodopád, iterativní vývoj, něco jiného?
- Jak moc modelovat a řídit se modelem, aneb agilní versus rigorózní přístup k vývoji softwaru.
- **Hlavní cíl dnešní přednášky:** udělat si pořádek v základních pojmech a konceptech

Strukturované vs. objektové modelování softwaru

Proč modelovat IS?

- Informační systémy jsou tvořeny **daty** a **operacemi**, které data zpracovávají a prezentují uživatelům
- Mnoho vazeb => složitý systém nelze zvládnout jako jeden celek
- Modelování = zvládnutí složitosti pomocí principu „rozděl a panuj“



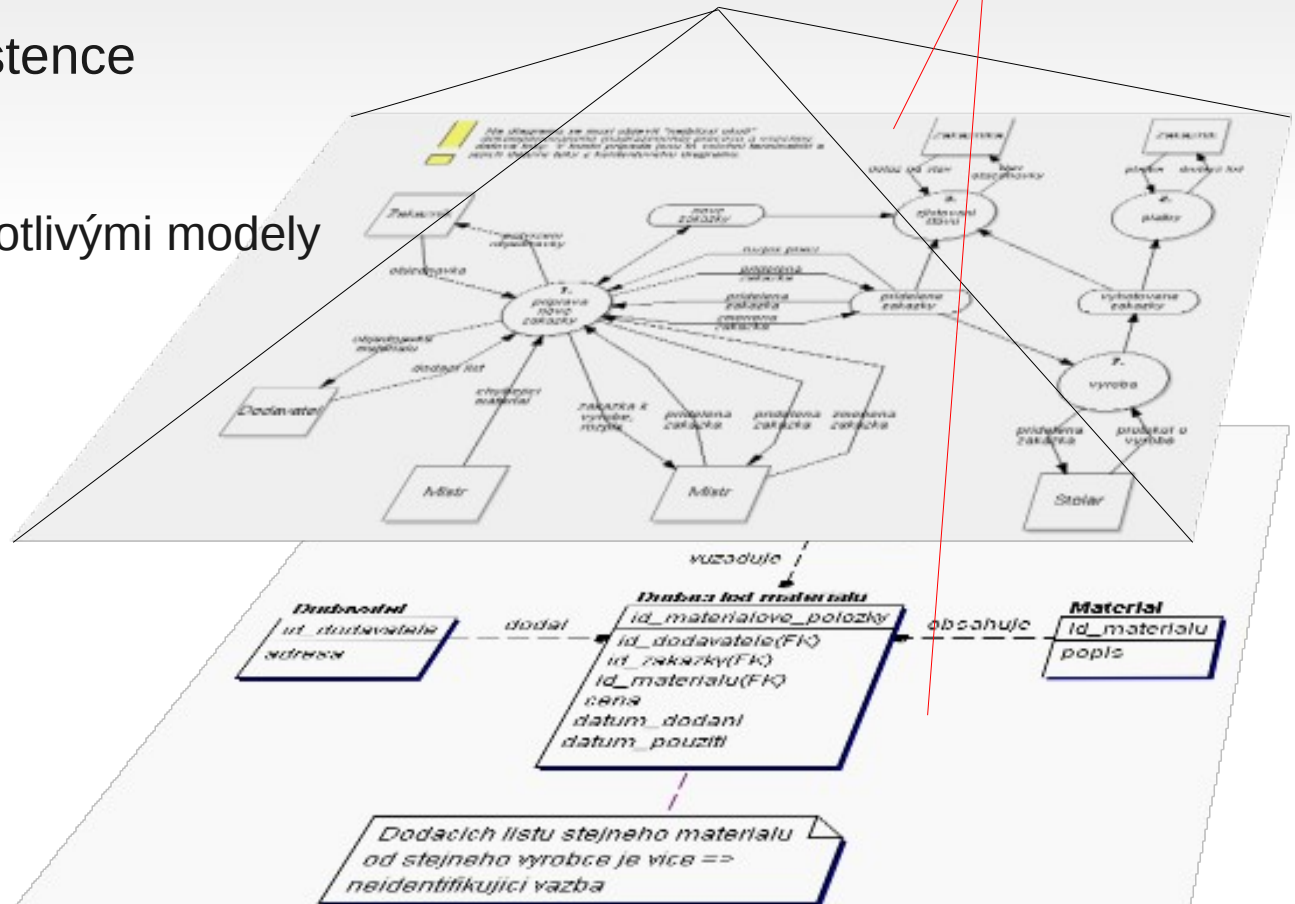
Zdroj: objekty.vse.cz

Strukturované modelování

- Oddělené funkční a datové modely
 - Kontextový diagram, DFD, události, ...
 - ERD, datový slovník, ...
- Postupné zpřesňování modelů
- Snaha o zachování konzistence
 - uvnitř modelu
 - vzájemně mezi jednotlivými modely

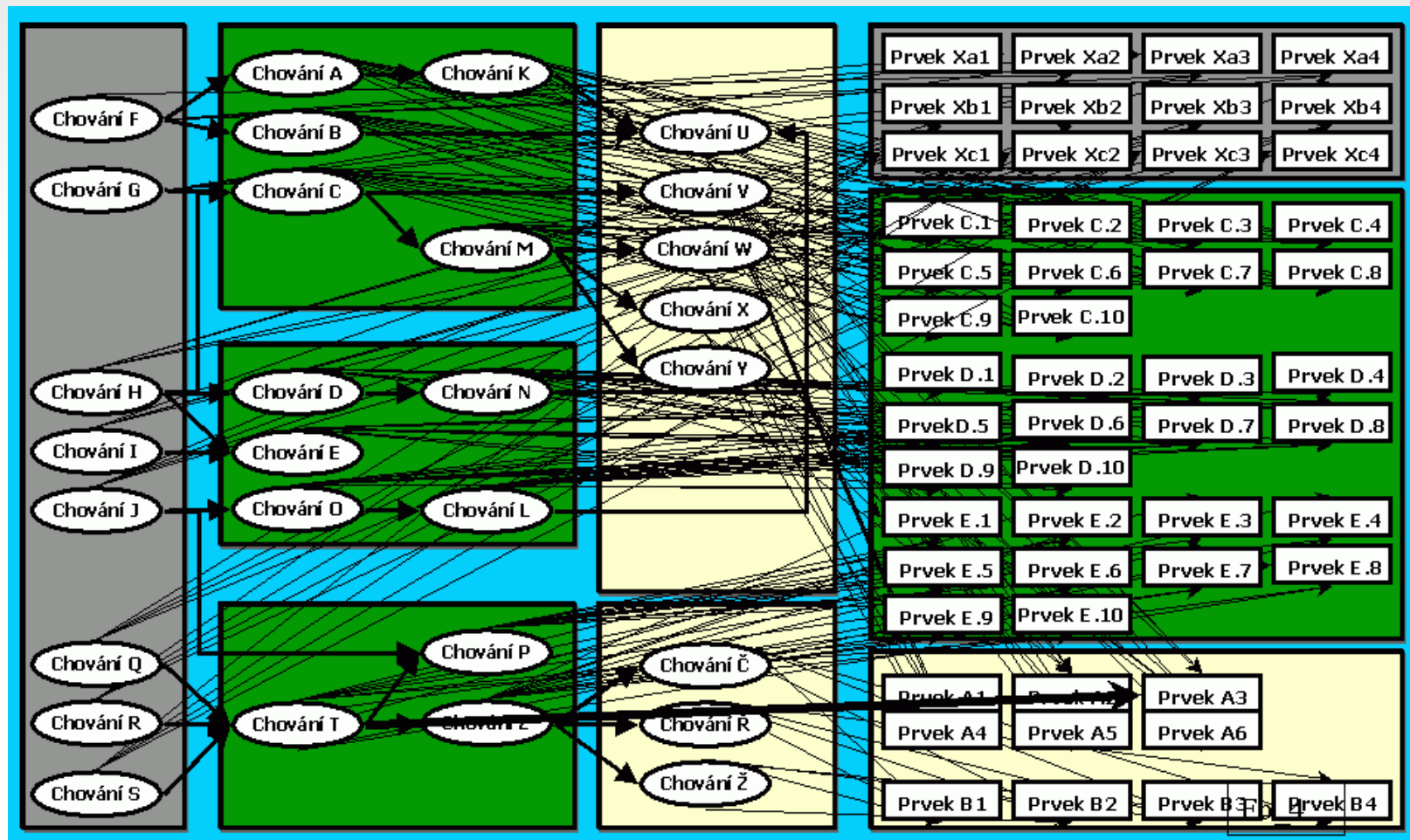
Vyvažování uvnitř modelu

Vyvažování modelů



Strukturované modelování pokr.

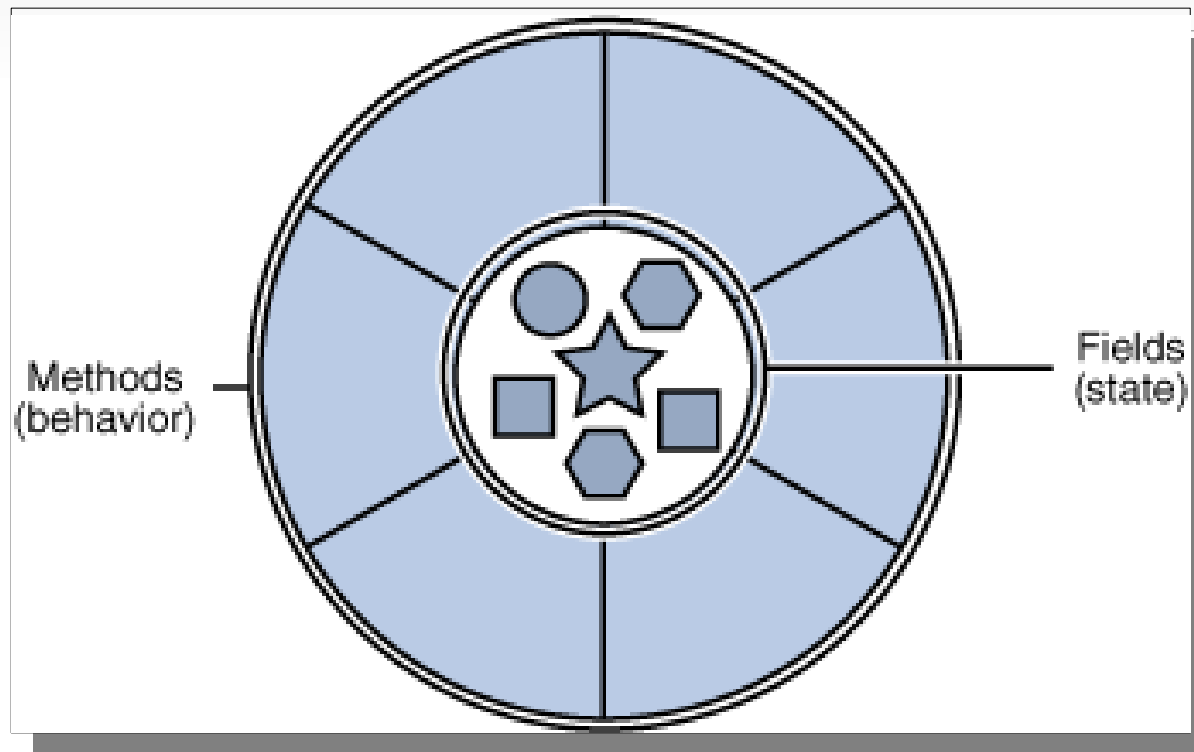
- Uspořádáním funkcí hierarchicky (funkční model) a dat (datový model)
- Usnadňuje orientaci ve funkčním a datovém modelu
- Stále velká složitost vztahů zejména mezi funkčním a datovým modelem



Zdroj: objekty.vse.cz

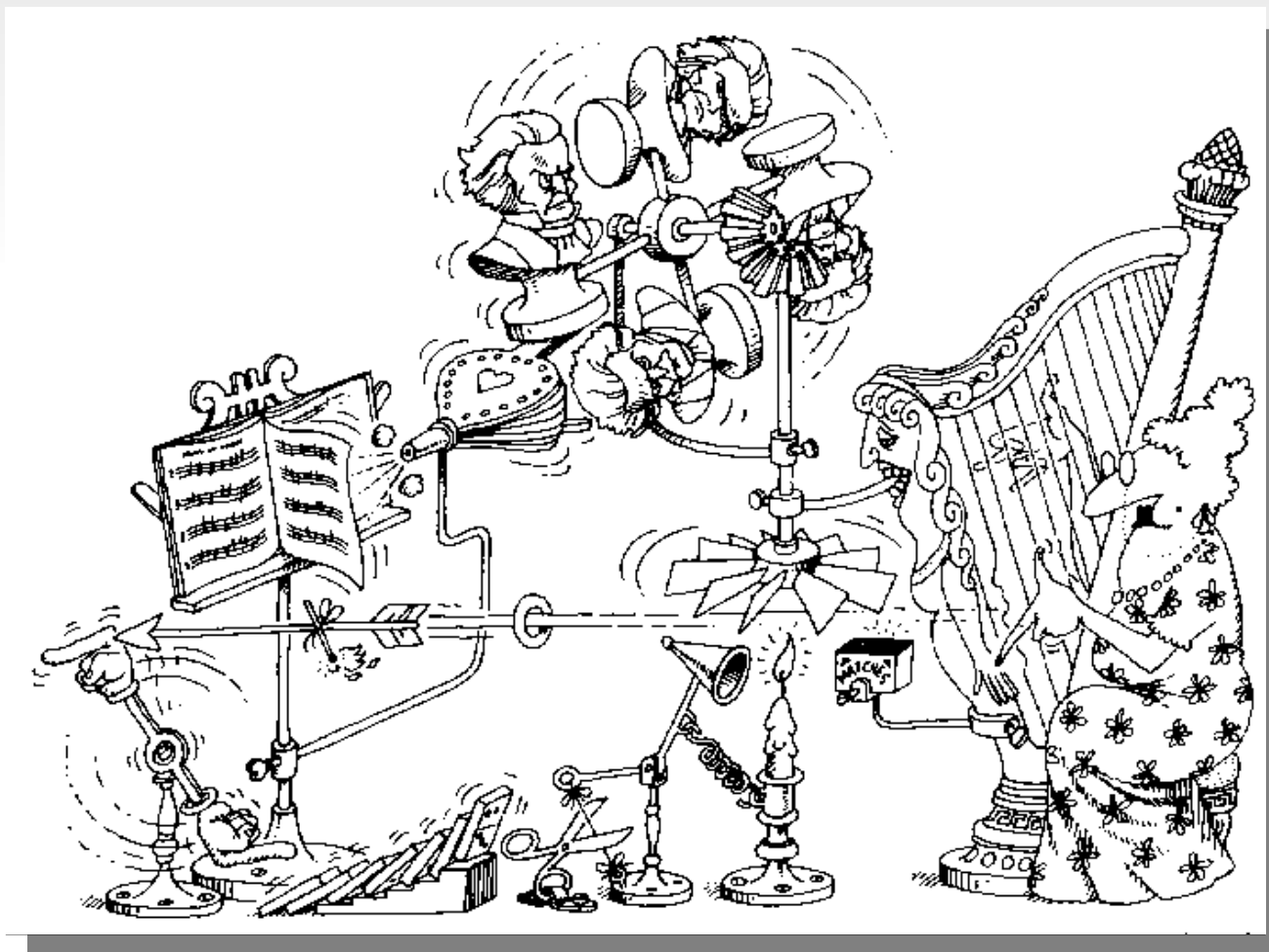
Co jsou to objekty

- Objekty kombinují data a funkce do podoby uzavřené, soudržné jednotky
- Objekty ukrývají data za vrstvou funkcí (operací)
 - Data jsou přístupná skrze operace
 - Zapouzdření
 - Stav, chování, identita



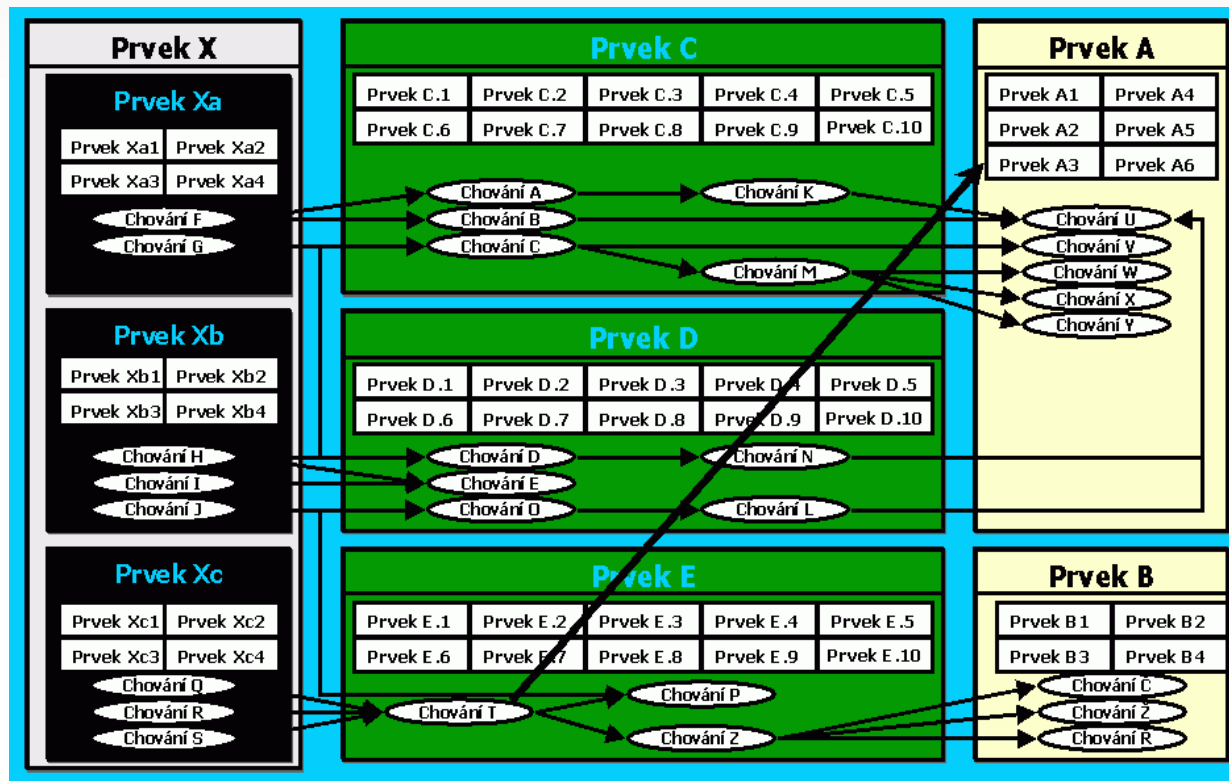
Spolupráce objektů, zasílání zpráv

- Objektový systém funguje na základě spolupráce jednotlivých objektů
- Objekty si posílají zprávy (vyvolávají operace na jiných objektech) a tím vytvářejí odezvu systému na požadavky uživatelů.



Objektové modelování

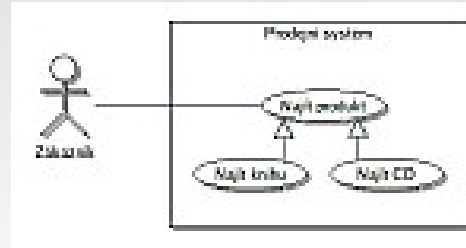
- Rozdělení systému na elementy (objekty), které v sobě zahrnují jak data, tak operace => závislosti mezi souvisejícími funkcemi a daty jsou vnitřní záležitostmi objektů
- Vztahy mezi objekty jsou mnohem jednodušší a jednoznačnější
- Uspořádání objektů do hierarchií ještě více zpřehledňuje systém
- OO modelování = rozdělení dat a funkcí do objektů a hierarchií



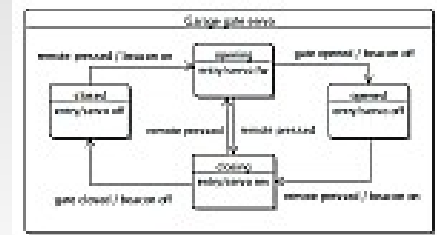
Objektové modelování pokr.

- Více modelů než u strukturovaného modelování
- Ne všechny modely se ale vždy používají a ne ve všech etapách nebo všech částech systému
- Postupné zpřesňování modelů
- Snaha o zachování konzistence
 - uvnitř modelu
 - vzájemně mezi jednotlivými modely
- **Hlavní cílový digram je diagram tříd. Ostatní diagramy slouží převážně k nalezení správného diagramu tříd**
- Iterativní a inkrementální vývoj

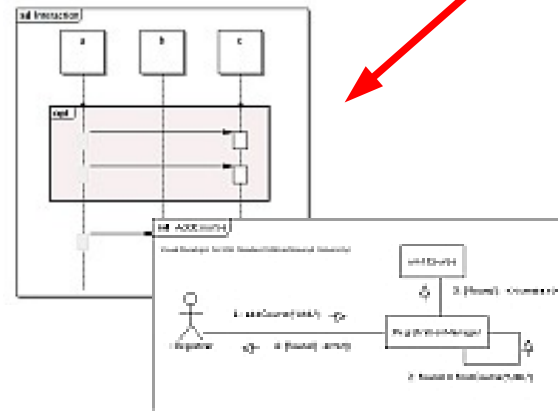
Případy užití



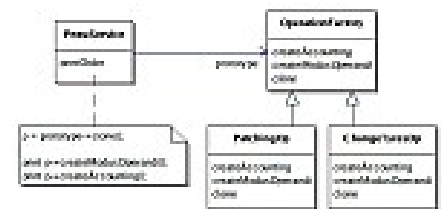
Stavové diag.



Diag. interakcí



Diag. objektů a tříd

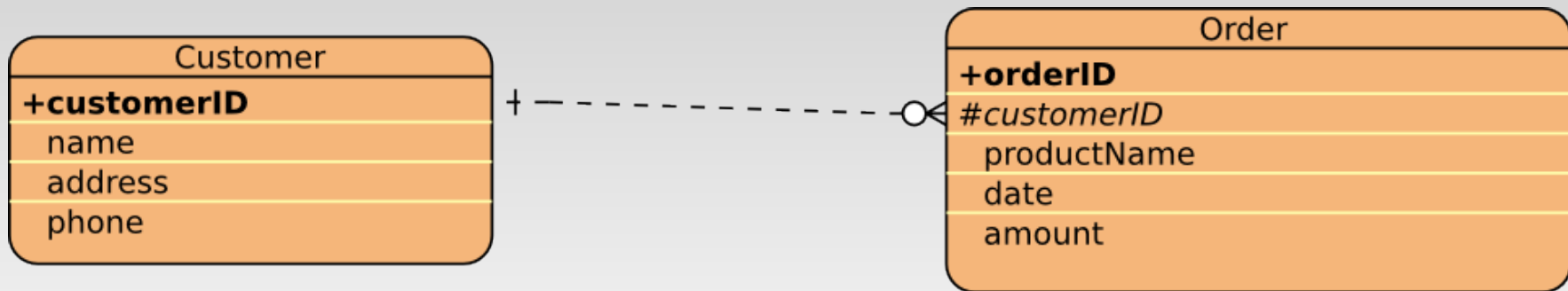


Vyvažování modelů

ORM: Object-Relational Mapping

- V reálném světě se často vyvíjí objektově, ale data se ukládají do relační databáze
 - Relační databáze je skvěle zvládnutá a časem prověřená (efektivita, škálovatelnost, kontrola integrity dat, ...)
- => Nutnost mapování objektů na entity
- => Object-Relational Mapping, ORM
 - Java Persistence API, Hibernate, ...

Relační databáze



■ Relační technologie

- Uložení dat v tabulkách
- Řádky jsou záznamy, sloupce typy
- Tabulky jsou propojeny vazbami
- Primární/cizí klíče
- Kardinalita/násobnost vazeb
- **Relační algebra** (SQL) pro snadný přístup k datům

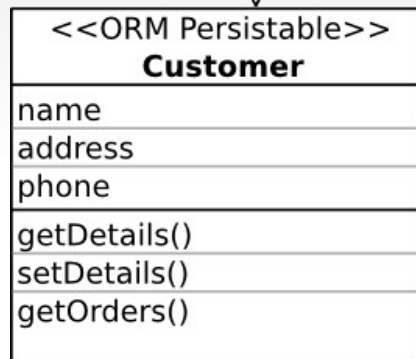
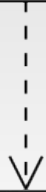
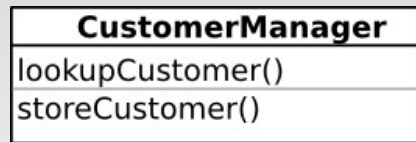
■ Objektová technologie

- Třídy obsahují data i operace
- Asociace s násobností
- Dědičnost
- Identita objektu není dána klíčem, ale často adresou
- „Asociace a objekty se udržují v paměti“ => operace nad daty jsou řešeny interakcí objektů
- Příklad: vyhledání všech studentů, kteří mají zapsaný daný kurz – rozdíl při použití SQL oproti interakci objektů

ORM – základní mapování

- Perzistentní třída definuje entitní množinu (tabulku)
- Objekt definuje entitu (záznam, řádek tabulky)
- Atributy třídy se stávají atributy entity (sloupce tabulky)
- Klíč je vybrán z atributů nebo je vytvořen nový
- Asociace/agregace/kompozice tříd definuje relaci (propojení tabulek cizími klíči)
- Dědičnost tříd
 - mapování 1:1
 - zahrnutí do nadtřídy
 - rozpuštění do podtříd

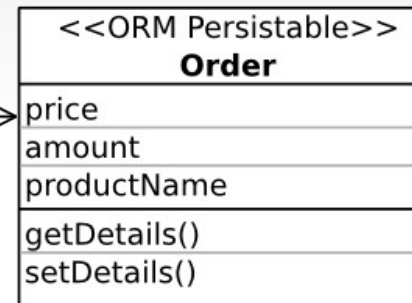
ORM – základní mapování (II)



1
X

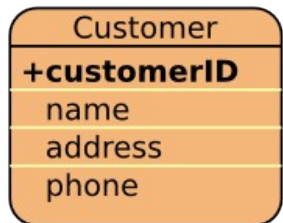
purchase

0..*
>

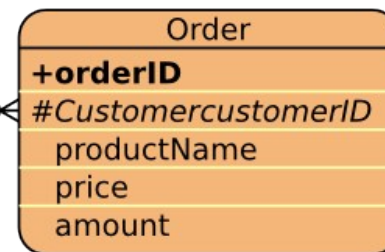


objekty pro práci s DB
(SQL dotazy)

perzistentní objekty



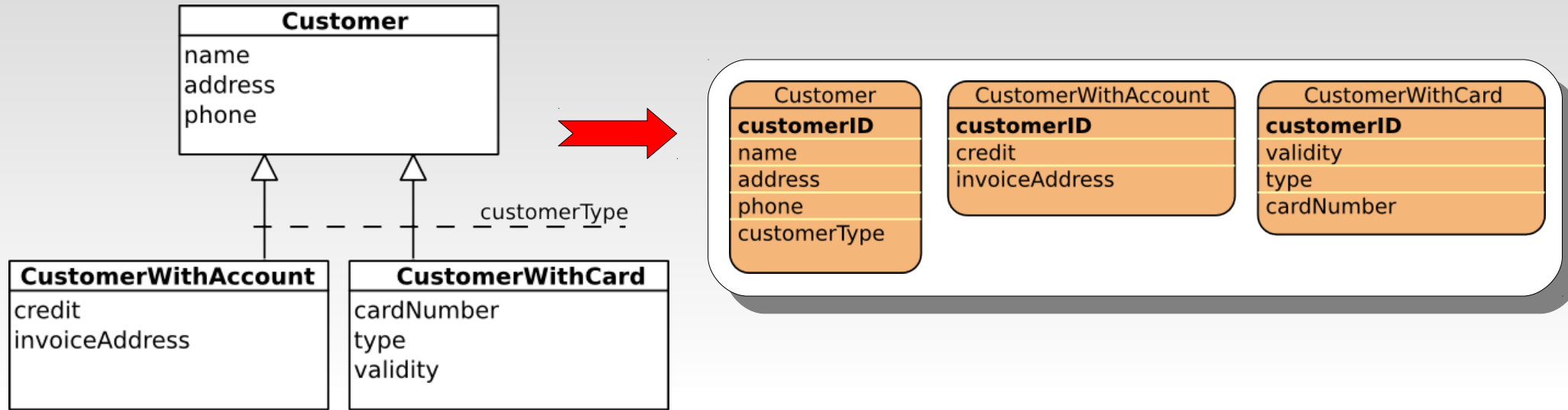
+



○

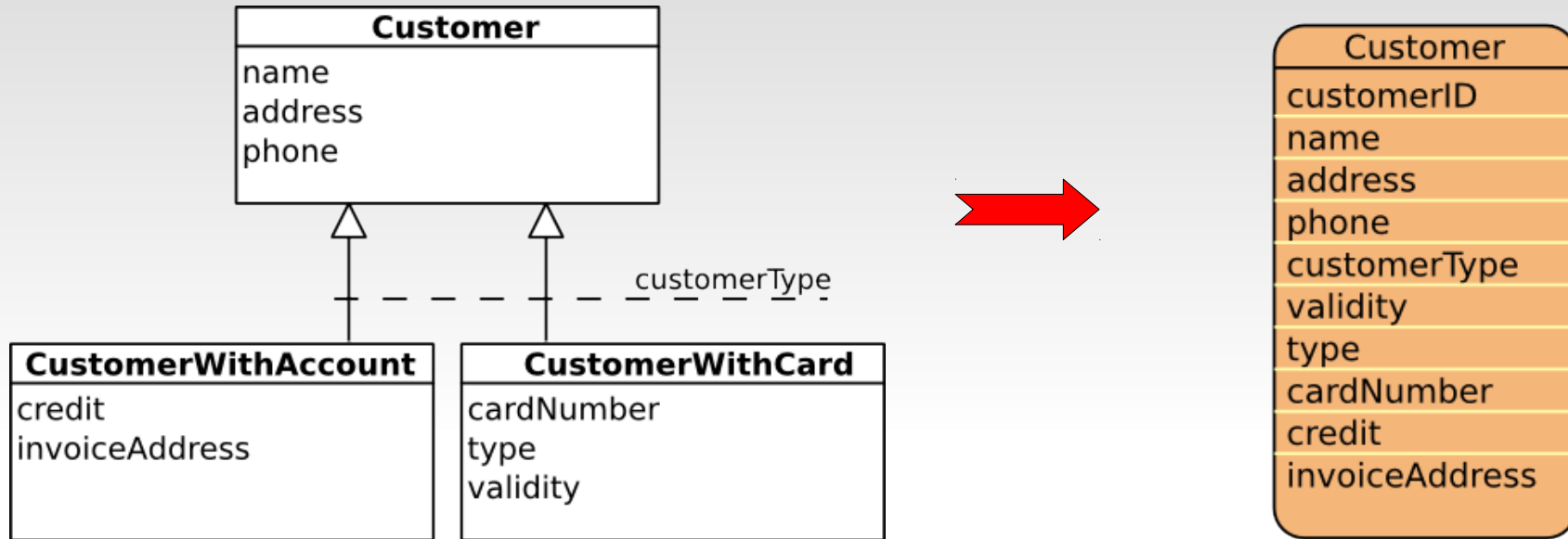
relační schéma

Dědičnost: mapování 1:1



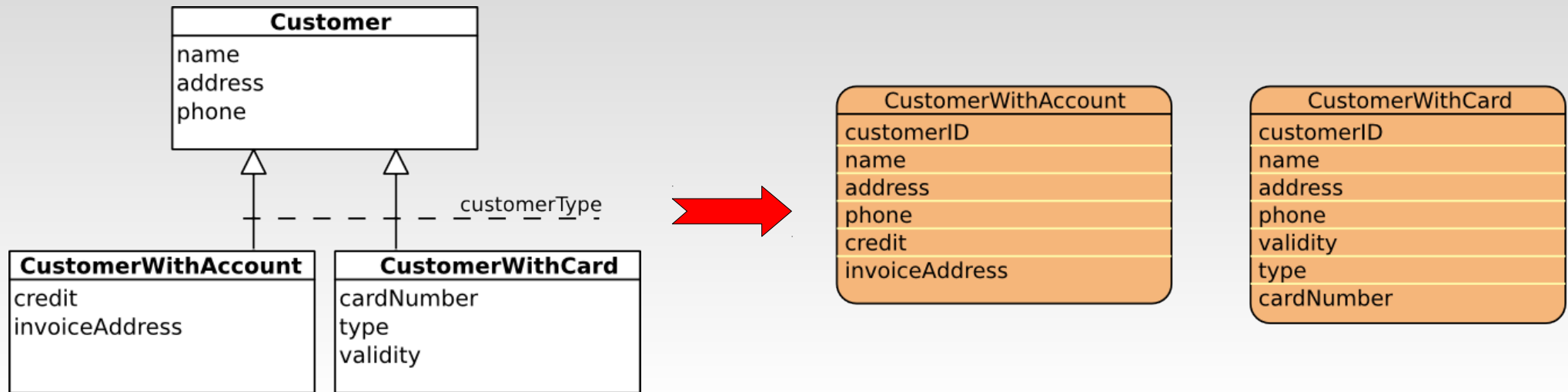
- Každá třída se stává tabulkou
- Všechny tabulky mají stejný primární klíč
- Diskriminátor se stává atributem
 - prohledávají se pouze tabulky příslušné podtřídy
- Jedna instance třídy je uložena v několika tabulkách
 - složitější přístup k datům

Dědičnost: zahrnutí do nadtřídy



- Všechny atributy podtříd jsou zahrnuty do jedné tabulky
- Některé atributy mohou obsahovat hodnotu NULL
 - porušení čtvrté normální formy
- Vhodné v případě menšího počtu podtříd s málo atributy

Dědičnost: rozpuštění do podtříd



- Atributy nadtřídy jsou přeneseny do tabulek pro všechny neabstraktní podtřídy
- Vhodné při:
 - Nadtřída má málo atributů
 - Existuje mnoho podtříd (košatý strom větvení)
 - Podtřídy mají hodně atributů

Životní cyklus softwaru

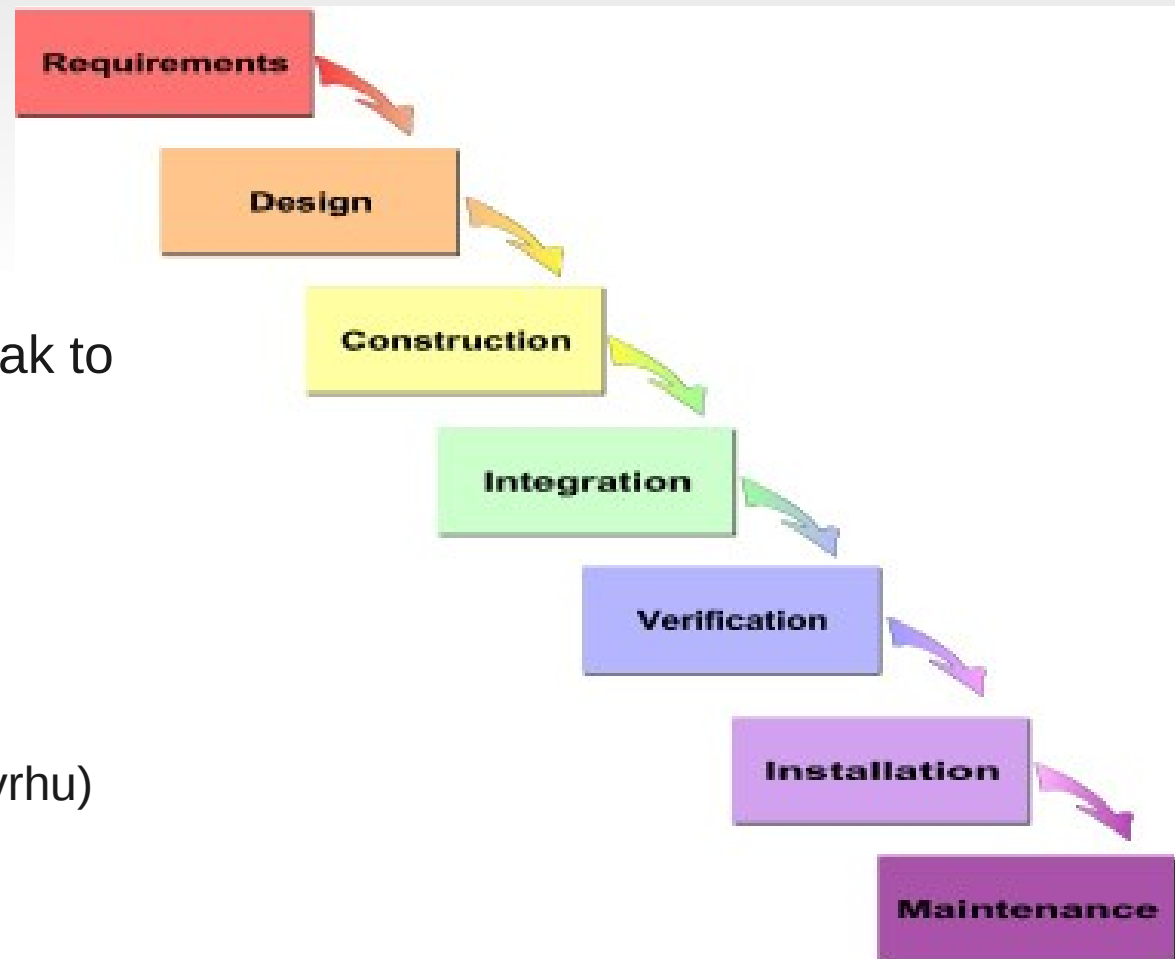
vodopád, iterativní a inkrementální postupy

Životní cyklus softwaru

- Životní cyklus SW popisuje život SW od jeho návrhu, přes implementaci, až po jeho předání a údržbu
- Existuje řada modelů:
 - Vodopád, letadlo, výzkumník, spirála, ...
- V praxi většinou používají dva modely:
 - Vodopád
 - Iterativní a inkrementální vývoj

Vývoj typu vodopád

- Rozděluje celý projekt na základě prováděných aktivit:
 - Analýza požadavků, návrh, kódování, testování apod.
- Příklad rozdělení ročního projektu:
 - 2 měsíce analýzy
 - 4 měsíce návrhu
 - 3 měsíce kódování
 - 3 měsíce testování
- Nikdy to není tak jednoduché, jak to na první pohled vypadá:
 - Chyby způsobují návrat do předchozí etapy/etap
 - Nejasná hranice mezi etapami
 - Překrývání etap (např. se implementuje ještě během návrhu)
 - Pozdní odhalení chyb
 - Odhad ceny



Vodopád – analogie s bytovým domem

- Vodopád = kompletní výstavba celého domu.
- Etapy vývoje:
 - Architekt kompletně navrhne dům, podzemní garáže, propozice bytů, ...
 - Projektant zhotoví všechny technické nákresy
 - Stavební firma vše postaví
 - Nájemci se nastěhují a bydlí
- Problém 1 (návrat do předchozí etapy)
 - Nájemník po čase zjistí, že střechou zatíká
 - Návrat na předchozí etapu => firma opraví střechu
 - Relativně levná oprava
- Problém 2 (návrat o několik etap zpět)
 - Nájemníci po nastěhování zjistí, že z kohoutků teče závadná voda z důvodu špatně navržených rozvodů vody.
 - Architekt navrhne nové řešení, projektant vyprojektuje, ...
 - Hodně drahá oprava.

Iterativní vývoj

- Iterativně = **předělávat**
- Celý projekt se opět díváme jako na jeden celek, ale jednotlivé části se postupně zpřesňují
- Iterace směřují k postupnému vylepšení, zpřesnění, doděláním nebo opravení částí systému
- Každá iterace obsahuje analýzu, návrh, testování apod. (tj. miniaturní vodopád), ale s různou intenzitou, např.:
 - V první iteraci provést celkovou analýzu požadavků a obrysový plán vývoje, více rozpracovat jádro systému, implementovat základní testovací třídy
 - V druhé iteraci podrobně rozpracovat důležité části systému, rozmodelovat je a částečně implementovat
 - Ve třetí iteraci podrobně rozpracovat méně podstatné části systému, doimplementovat věci z předchozí iterace
 - ...
- Vývoj typu vodopád je tedy iterativní vývoj s jedinou iterací
 - Analýza požadavků, analýza, návrh, kování i testování se provádí pouze jednou, výsledkem je hotový systém

Iterativní vývoj – analogie s bytovým domem

- První iterace:
 - Architekt navrhne vzhled a základní propozice domu a bytů a vzhled okolí domu.
 - Projektant rozpracuje technické nákresy domu.
 - Stavební firma postaví hrubou stavbu.
- Druhá iterace:
 - Architekt rozpracuje propozice bytů dle požadavků prvních nájemců.
 - Projektant vytvoří technické nákresy prvních bytů.
 - Firma dodělá společné prostory domu (omítky apod.) a vybuduje první byty.
 - Nastěhují se první nájemci.
- Třetí iterace:
 - Projektant vytvoří technické nákresy zbylých bytů
 - Stavební firma vybuduje zbylé byty a provede terénní úpravy kolem domu.
 - Stěhují se další nájemci.

=> rychlejší (dílčí) výsledky – první nájemníci dříve bydlí, byt' provizorně

=> rychlejší odhalení chyb – již první nájemníci zjistí problém s vodou

Inkrementální vývoj

- Inkrementálně = **přidávat k**
- Uplatňuje se zejména u větších projektů nebo v agilním vývoji
 - Několik velkých přírůstků – rigorózní vývoj, velké projekty
 - Mnoho malých přírůstků – agilní vývoj, průběžná integrace
- Jednotlivé části systému (přírůstky, inkrementy) vytváříme „nezávisle“ na zbytku a pak integrujeme
 - Nicméně i zde je potřeba mít alespoň částečný přehled o celém výsledném systému.
- Vývoj jednotlivých přírůstků může probíhat iterativně, vodopádem, XP, ...
 - Nejčastěji se používá iterativní vývoj přírůstků, často se mluví o iterativně-inkrementálním vývoji
- Vývoj typu vodopád je tedy inkrementální vývoj s jediným přírůstkem představujícím celý systém

Inkrementální vývoj – analogie s domem

- Mnoho malých přírůstků: např. stavba bytového domu po jednotlivých bytech. Zatímco horní patra ještě neexistují, dole se už bydlí.
 - Integrace = napojení nových bytů (rozvody apod.) na již existující infrastrukturu.
- Několik velkých přírůstků: např. stavba bytového komplexu o více budovách. Jeden přírůstek = jeden bytový dům.
 - Integrace = napojení nového bytového domu na stávající infrastrukturu (rozvody elektřiny, chodníky, ...)

=> Výstupem je vždy hotový funkční systém. Přírůstky se mohou řešit různě, nejčastěji se ale používá iterativní vývoj

Vodopád vs. iterativní/inkrementální vývoj

- Vodopád je mnohými analytiky z OO komunity považován za překonaný
 - Je těžké určit, jestli je projekt na správné cestě k úplné implementaci
 - Je příliš snadné prohlásit úvodní fáze za dokončené a přitom přehlédnout mnoho chyb
 - Jedinou jistotu, že je navržený software správný, nám dá až testování jeho implementace. Proto je lepší implementovat a testovat průběžně
- V praxi je vodopád velice často používán, přestože firma deklaruje iterativní/inkrementální vývoj :-(
 - *„Děláme jednu analytickou iteraci následovanou dvěma návrhovými iteracemi...”*
 - Skutečná iterace/inkrement produkuje otestovaný kód jehož kvalita se co nejvíce blíží kvalitě finálního produktu
 - *„Kód této iterace má spoustu chyb, ale opravíme je až potom.”*
 - Testování a integrace jsou velmi náročné aktivity, které by rozhodně neměly zůstat nedokončené

Vlastnosti iterativního a inkrement. vývoje

- Průběžná implementace a testování
 - včasné varování o chybách
- Nutnost předělávat kód
 - Předpokládá se, že pozdější iterace nebo přírůstky budou upravovat nebo mazat existující kód
 - Ve vývoji SW to nemusí být až tak velká nevýhoda – je lepší špatný kód přepsat, než ho nějak obcházet
 - Existují techniky, které zefektivňují úpravy kódu
 - Automatické testování (*regression tests, unit tests*)
 - testovací třídy, např. JUnit pro Javu
 - RefaktORIZACE (*refactoring*)
 - pravidla pro transformaci zdrojových kódů, může být zautomatizováno
 - Průběžná integrace (*continuous integration*)
 - např. automatický překlad kódu při uložení změn programátorem + automatické testování

Iterativní a inkrementální vývoj používejte pouze pro projekty,
se kterými chcete uspět :-)

-- Martin Fowler: UML Distilled

Agilní vs. rigorózní vývoj

Agilní metody vývoje

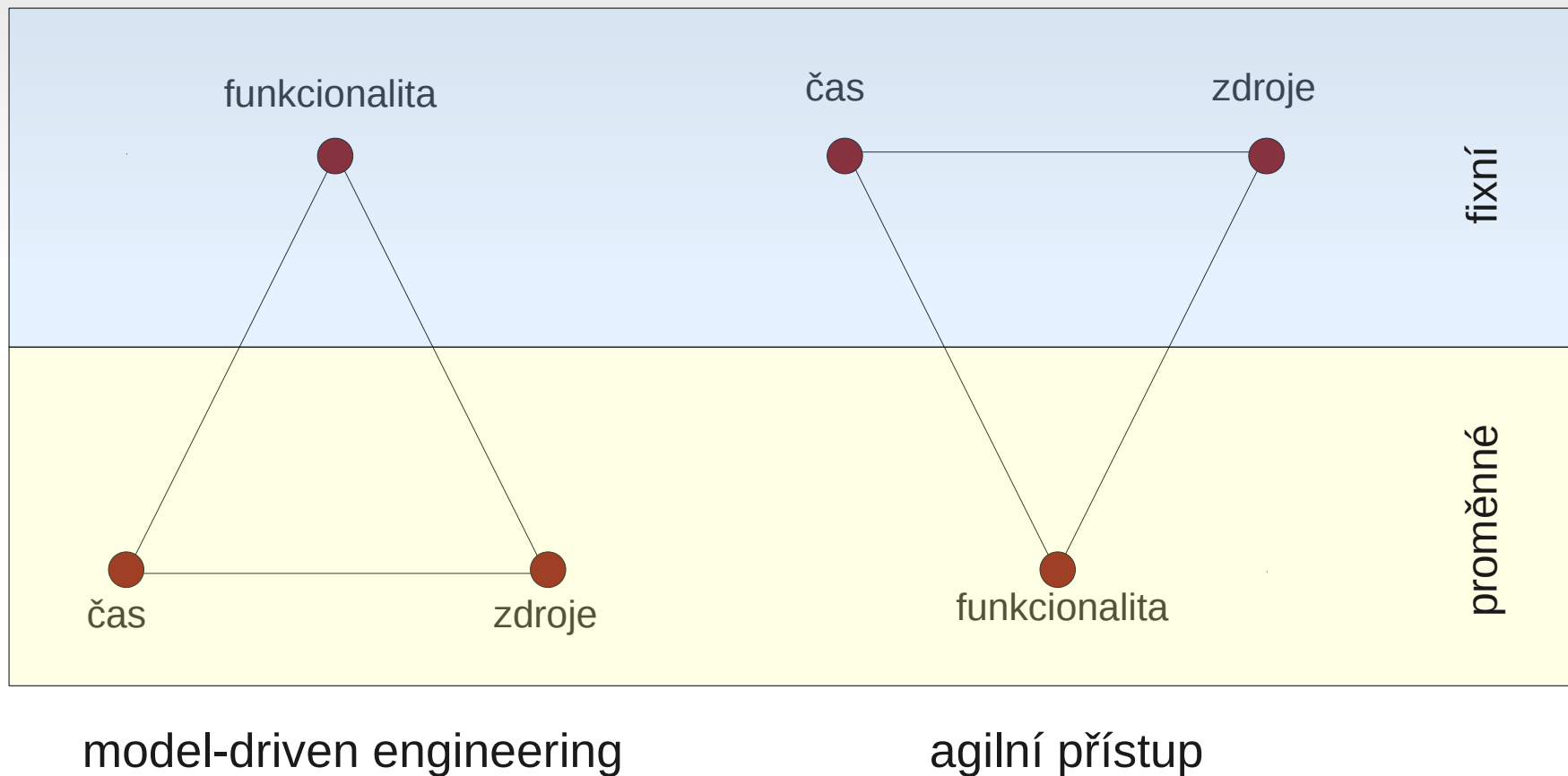
- *Agile development*
- Metody zaměřené více na lidi a vývojové procesy – určujícím faktorem v úspěchu projektu je kvalita lidí pracujících na projektu a jejich spolupráce
- Velmi krátké iterace (jeden měsíc a méně)
- Malé ale výkonné týmy (dvojice)
- **UML se používá pouze jako doplněk**
- Zapojení zákazníka do vývoje (zákazník se účastní sestavování návrhu a testů, ideálně je součástí vývojového týmu)
- Agilní metodiky: Extreme Programming (XP), Scrum, Feature Driven Development (FDD), Crystal, Dynamic Systems Development Method (DSDM), ...
- Manifesto of Agile Software Development <http://agileManifesto.org>
 - Individuality a interakce mají přednost před nástroji a procesy
 - Fungující software má přednost před obsáhlou dokumentací
 - Spolupráce se zákazníkem má přednost před sjednáváním smluv
 - Reakce na změnu má přednost před plněním plánu

Rigorózní = vývoj řízený modely

- Mnoho akronymů:
 - *Model-Driven Engineering, MDE*
 - *Model-Driven Development, MDD*
 - *Model-Driven Software Engineering, MDSE*
 - *Model-Driven Software Development, MDSD*
- „Seriózní“ přístup, kdy implementujeme striktně na základě UML modelů
 - Průběžně vzniká dokumentace
 - Podpora pro fázi nasazení a údržby
- **UML je základním nástrojem MDE**
- Metodiky Iconix, Select Perspective, ...

MDE vs. agilní přístup

- Tři provázané proměnné důležité pro řízení SW projektů: požadovaná funkcionality, dostupné zdroje a čas



RUP

Rational Unified Process

(Rational) Unified Process

- Unified Process (UP)
 - Jacobson, Booch, Rumbaugh
 - otevřený, obecný koncept
- Rational Unified Process (RUP)
 - komerční licencovaná verze UP od IBM
 - rozšiřuje UP
 - liší se v detailech, UP a RUP se dají pokládat za synonyma
 - UP i RUP mají mnohem více společného než rozdílného
- Nejedná se přímo o metodiku (postup), ale spíše o „*process framework*“
 - Nejdříve je třeba stanovit vhodný postup pro konkrétní projekt (tzv. *development case*)
 - Často se přizpůsobuje i firmě a nástrojům

- **Role, pracovník** (*worker*)

- modeluje „kdo“ v procesu vývoje softwaru
- role v projektu pro jednotlivce nebo tým
- každý *worker* může být realizovaný několika jednotlivci nebo týmy a každý jedinec nebo tým může vystupovat v několika *workers*.
- v RUPu se používá termín *role* namísto *worker*, význam je ale stejný



- **Aktivita** (*activity*)

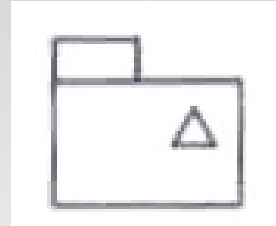
- modeluje „co“ v procesu vývoje softwaru
- úkol v projektu vykonávaný jednotlivci nebo týmy
- jednotlivci a týmy musí při provádění aktivity vždy přijmout UP i RUP proto s aktivitami asociuje role (workers)
- aktivity mohou být dekomponovány na podrobnější úrovně



Notace RUP pokr.

- **Artefakt** (*artifact*)

- modeluje „co“ v procesu vývoje softwaru
- vstupy a výstupy projektu
 - zdrojový kód, spustitelný program, standardy, dokumentace, ...
- mohou být znázorněny různými ikonami v závislosti na tom, co představují







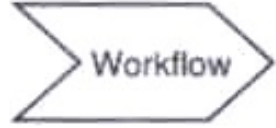




- **Tok práce** (*workflow*)

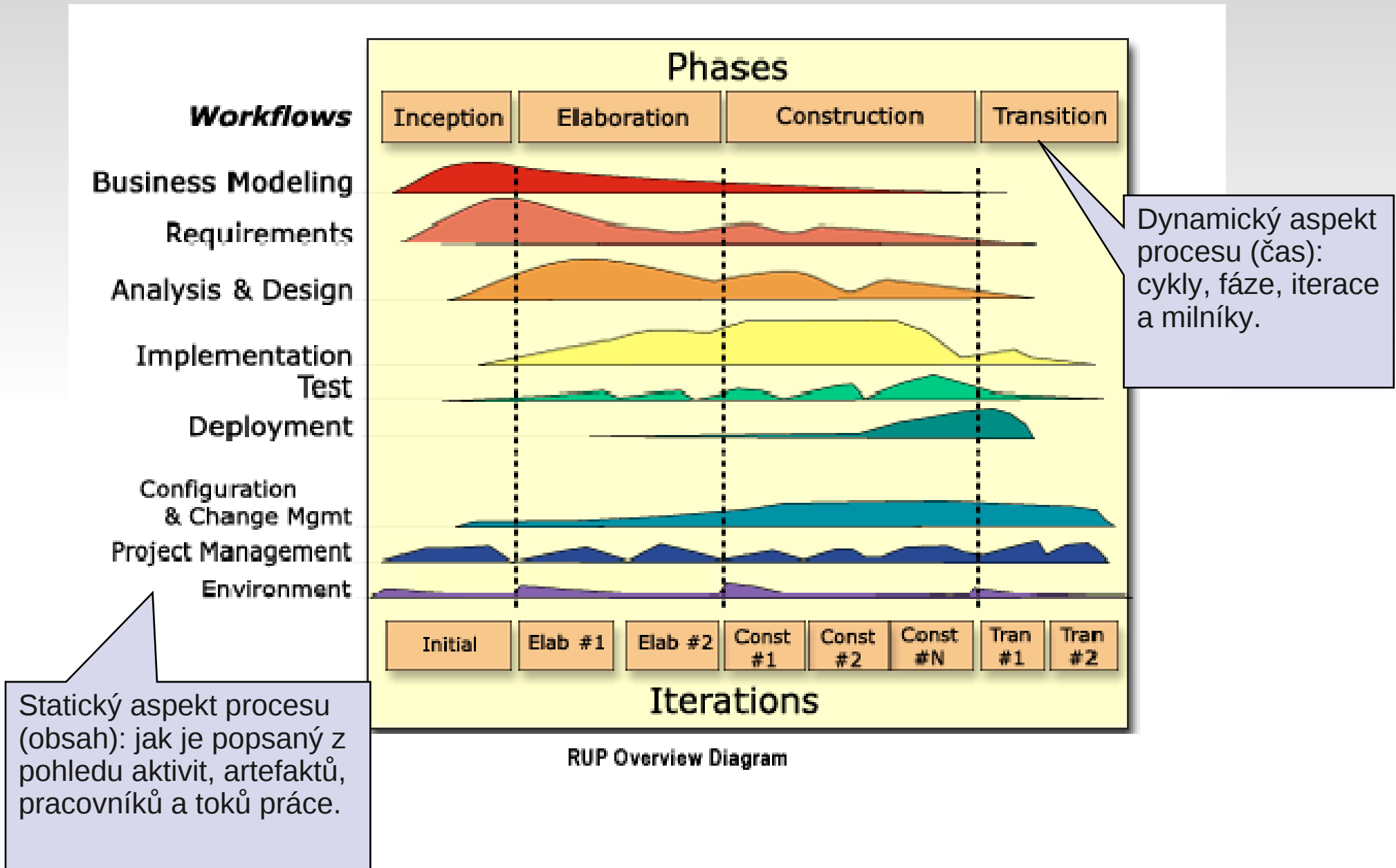
- modeluje „kdy“ v procesu vývoje softwaru
- sekvence aktivit vykonávaných pracovníky (workers)
- mohou být dekomponovány na jeden a více detailnějších modelů
- detailnější toky práce jsou pouze odkazovány ikonou



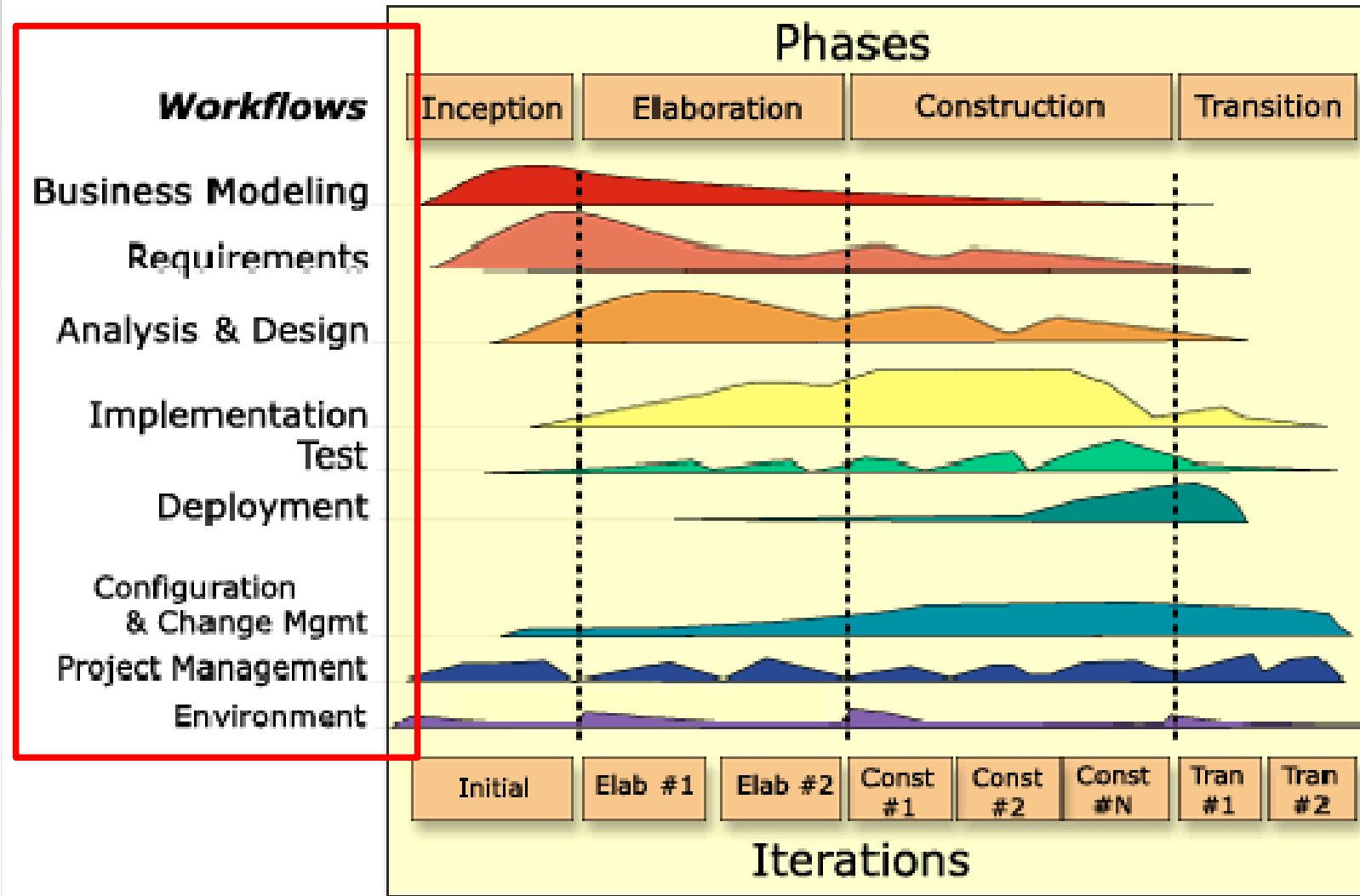
Notace UP vs. RUP

UP	RUP	Semantics
 Worker	 Role	<i>Who</i> – A role in the project played by an individual or team
 Activity  Artifact	 Activity  Artifact	<i>What</i> – A unit of work performed by a worker (role) or an artifact produced in the project
 Workflow Workflow Detail	 Discipline  Workflow Detail	<i>When</i> – A sequence of related activities that brings value to the project

RUP: Dvě dimenze vývoje softwaru



RUP: Workflows



RUP Overview Diagram

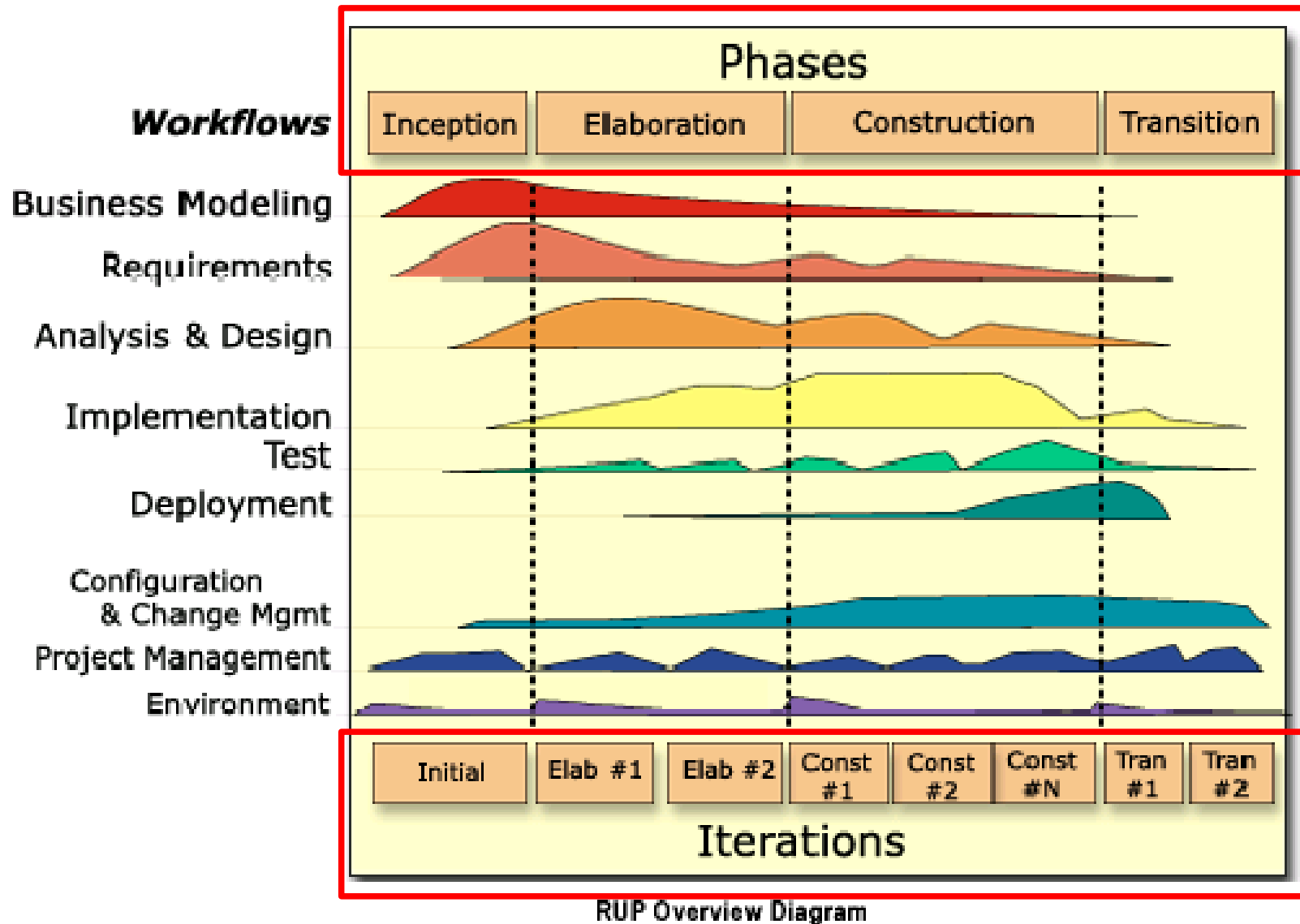
RUP: Workflows (I)

- Business modeling
 - Modelování činností uvnitř firmy, která bude navrhovaný IS používat
- Requirements
 - Zachycení uživatelských požadavků
 - Stanovení hranice systému, později její zpřesňování
 - Modelování základních use-casů, v pozdějších iteracích jejich rozpracování
 - Dokumentace use-casů, nejprve jednou větou, později scénáře, toky událostí, apod.
- Analysis and Design
 - Nalezení základních objektů, tříd a balíčků, později jejich rozpracování do podoby návrhových tříd, rozhraní, komponent, ...
 - Využití analytických a návrhových vzorů

RUP: Workflows (II)

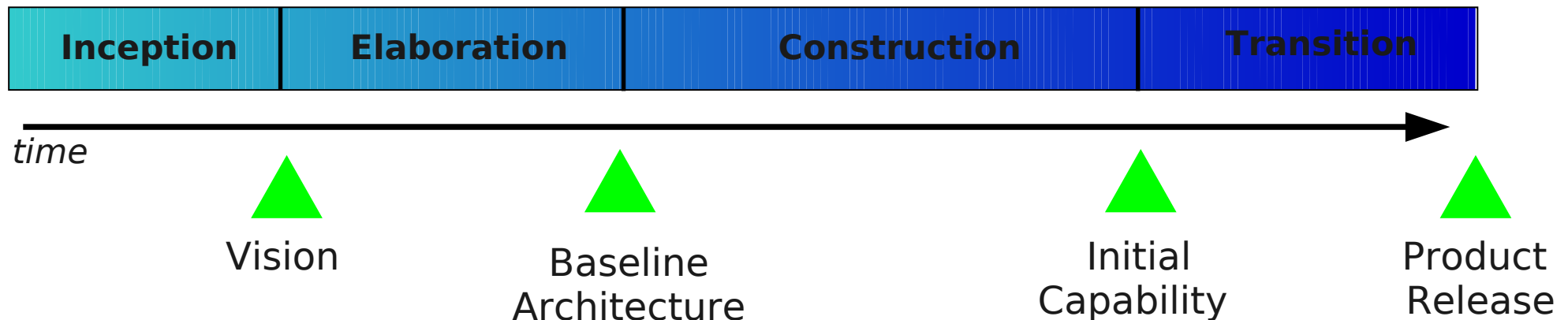
- Implementation
 - Modely rozmístění (deployment models)
 - Implementace (kódování)
- Test
 - Jednotkové testy (Unit Testing)
 - Integrační testy (Integration Testing)
 - Systémové testy (System Testing)
- Deployment
 - Produkční nasazení
- Configuration and Change Management
 - Zachycení požadavků na změny od různých zdrojů (nové požadavky od uživatelů, opravy chyb od testerů, ...)
 - Rozdělení do subsystémů pro jednotlivé týmy
 - Stanovení „secure workspaces“

RUP: Cykly, fáze, iterace



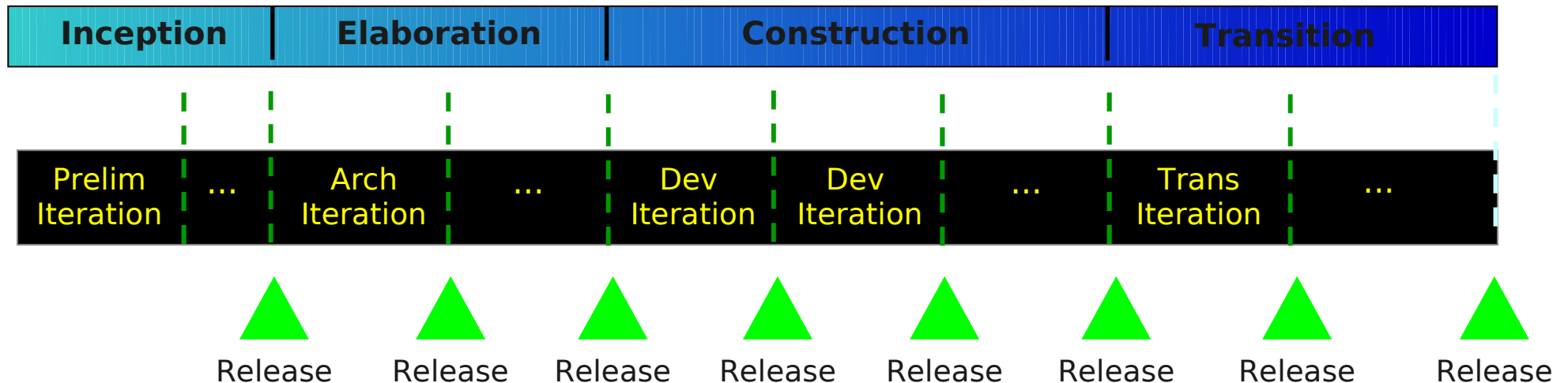
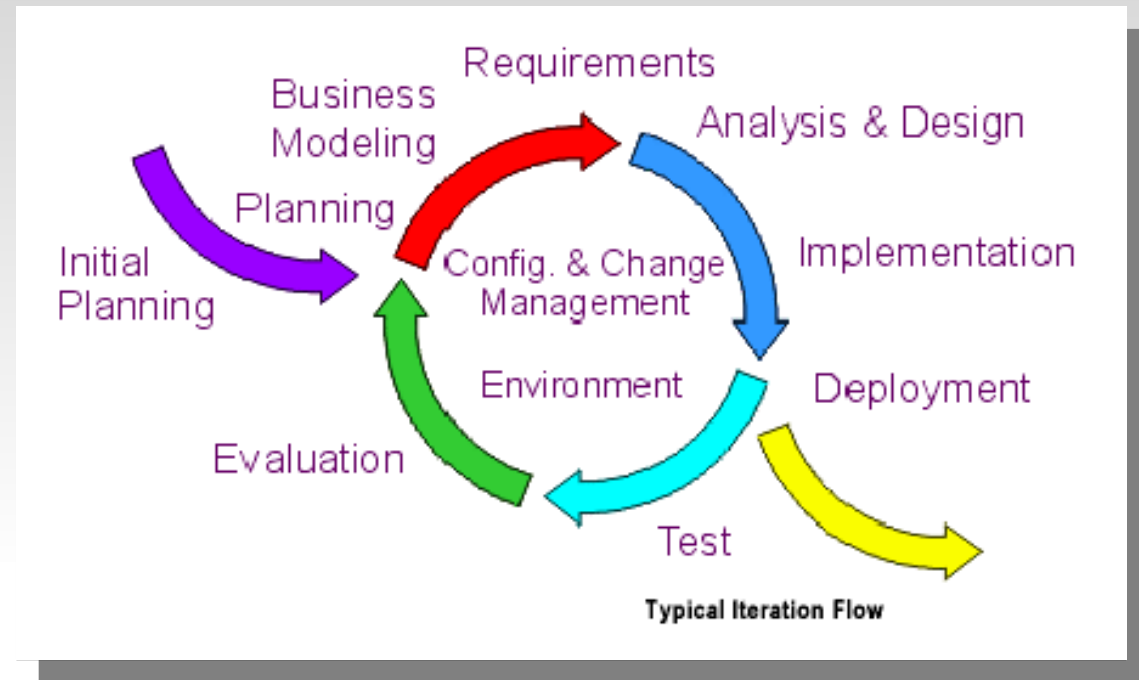
RUP: Cykly, fáze, iterace (I)

- Výrobek prochází různými **cykly** (= inkrementy v inkrementálním vývoji)
 - každý cyklus produkuje novou verzi systému (release)
- Každý cyklus se skládá z těchto **fází**:
 - **Zahájení (Inception)**: potvrzení konceptu, proveditelnost, cíle
 - **Rozpracování (Elaboration)**: detailní požadavky, scénáře použití, architektura, komponenty, vzory interakce na vysoké úrovni
 - **Konstrukce (Construction)**: zdokonalení architektury, implementační iterace řízené rizikem
 - **Předání (Transition)**: uživatelské přijetí, ...



Iterace

- Každá fáze může být dále rozdělena do různých **iterací**.
- Každá iterace obsahuje **analýzu, návrh a implementační** aktivity.
- Každá iterace produkuje spustitelnou verzi (release), buďto interní nebo externí



Fáze zahájení (Inception phase)

■ Cíl

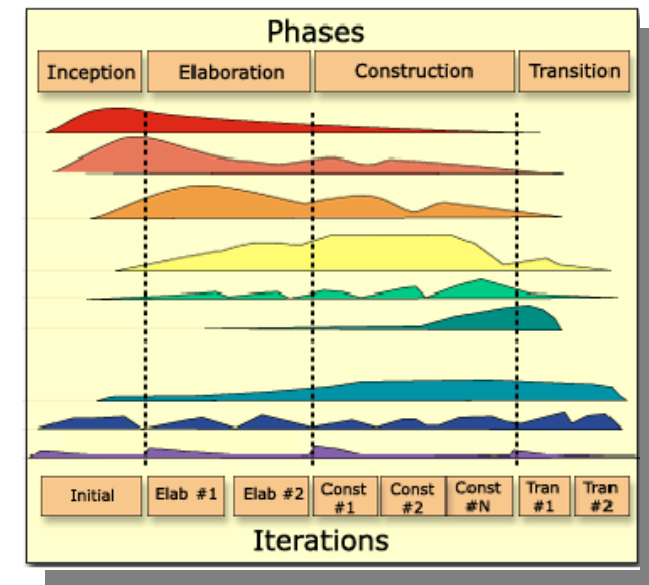
- ustanovení obchodního případu u nového systému
- specifikace rozsahu

■ Výstup

- obecný přehled o požadavcích na projekt, tj. klíčové požadavky
 - iniciální model případů užití a iniciální model domény (hotovo z 10-20%)
 - popis případů užití a aktérů jednou větou
- iniciální obchodní případ, včetně:
 - kritérií úspěšnosti
 - iniciální posouzení rizik
 - odhad potřebných zdrojů

■ Milník

- definice cílů životního cyklu



Fáze rozpracování (Elaboration Phase)

■ Cíl

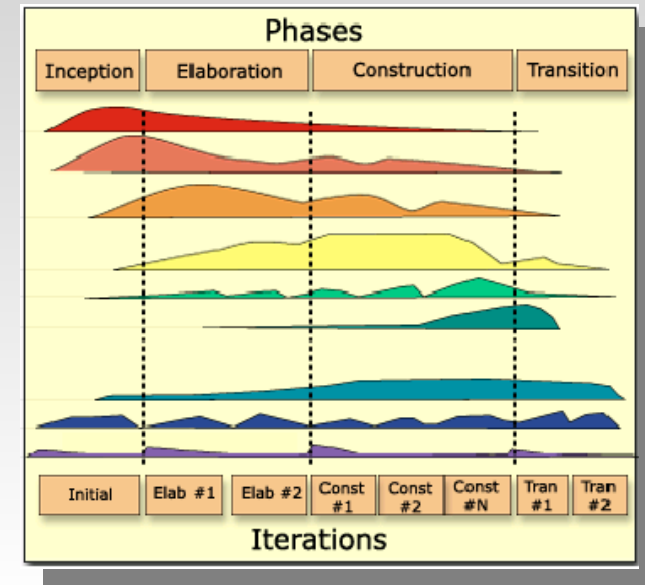
- analyzovat problémovou oblast
- ustanovit základy architektury
- určit nejrizikovější části projektu
- vytvořit kompletní plán pro dokončení projektu

■ Výstup

- model případů užití a model domény z 80% hotové
 - seznam aktérů, dokumentace případů užití
- realizovatelná architektura a její průvodní dokumentace
- aktualizovaný obchodní případ, vč. aktualizace posouzení rizik
- plán vývoje celého projektu
 - rozdělený do iterací, odhad ceny pro jednotlivé iterace

■ Milník

- architektura životního cyklu



Fáze konstrukce (Construction Phase)

■ Cíl

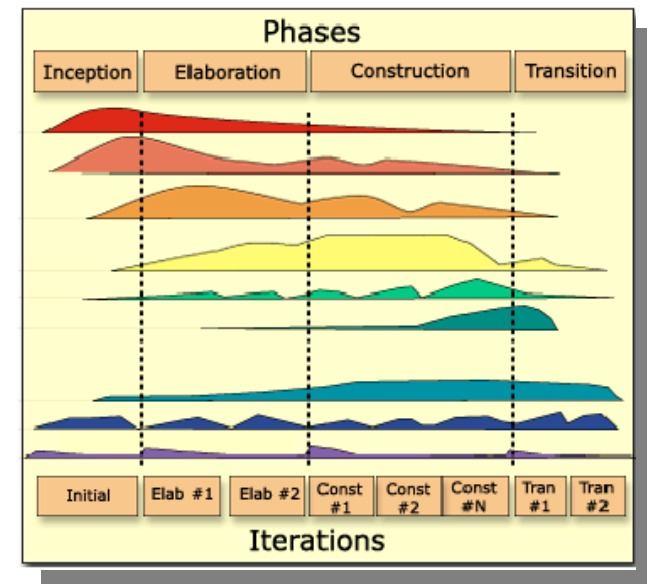
- inkrementálně vyvíjet a dokončit softwarový produkt, který bude připravený k předání zákazníkovi

■ Výstup

- kompletní model případů užití a návrhový model tříd
- spustitelné verze s novými funkcemi
- uživatelská dokumentace
- dokumentace nasazení (administrace)
- hodnotící kritéria pro každou iteraci
- popisy verzí
- aktualizovaný vývojový plán

■ Milník

- iniciální provozuschopný systém



Fáze předání (Transition Phase)

- **Cíl**

- předat software uživatelům

- **Výstup**

- spustitelné verze
- aktualizovaný model systému
- hodnotící kritéria pro každou iteraci
- popisy verzí
- aktualizovaná uživatelská dokumentace
- aktualizovaná administrátorská dokumentace

- **Milník**

- nová verze (product release)

