Strategie vývoje softwaru

objektové modelování, iterativní/inkrementální vývoj, agilní a model-driven 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**

Doporučená literatura:

- Arlow, Jim - Neustadt, Ila: UML 2.0 and the unified process – practical object-oriented analysis and design. 2nd ed. Boston : Addison-Wesley, 2005.
- ...

Organizační záležitosti – zkouška

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

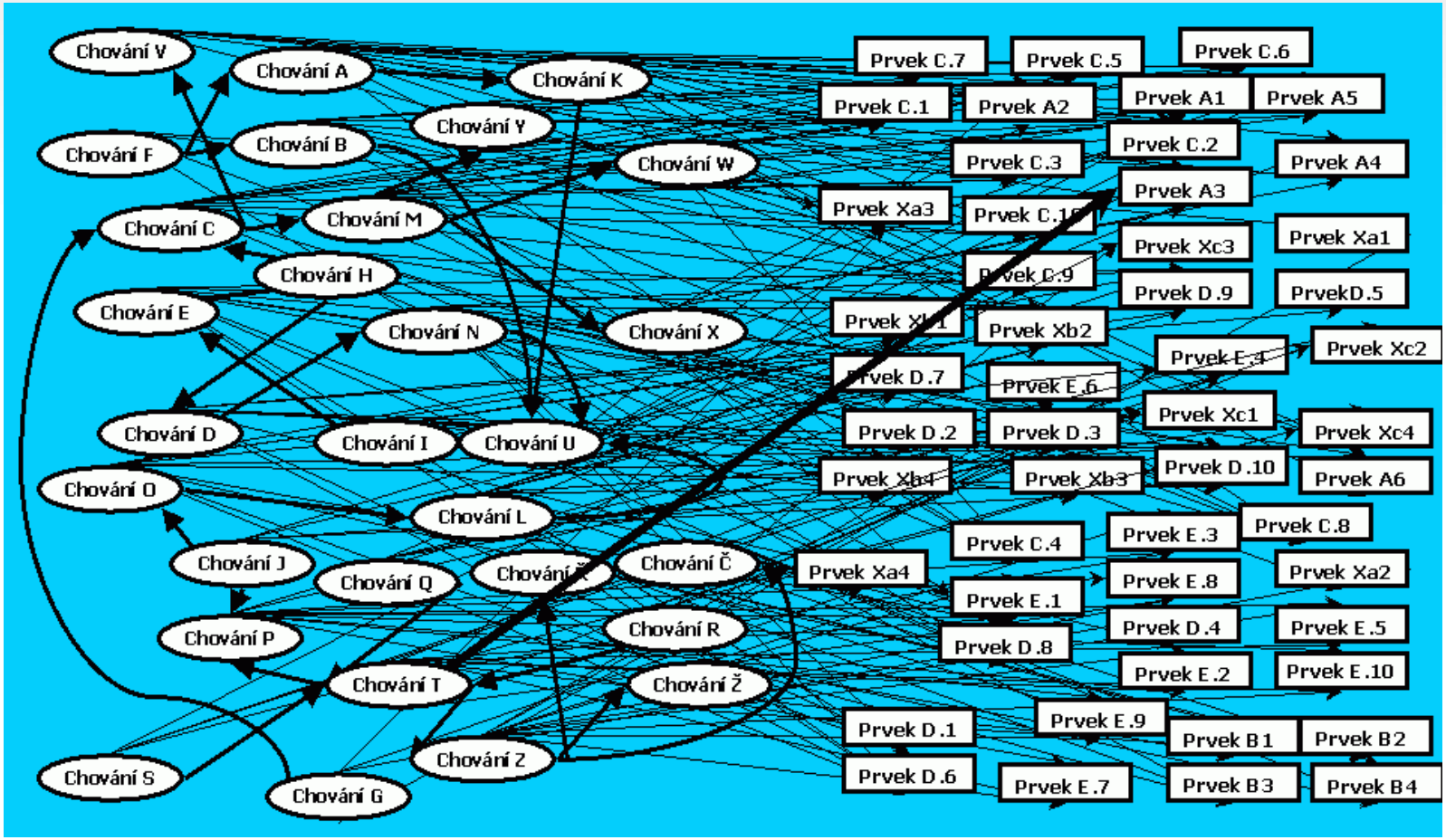
- Proč vůbec modelovat? Modelovat strukturovaně nebo objektově?
- Jaký použít model životního cyklu, tj. návaznost činností při modelování? Vodopád, iterativní vývoj, něco jiného?
- Jak moc modelovat a řídit se modelem, aneb agilní versus model-driven přístup k vývoji softwaru.

- Hlavní cíl dnešní přednášky: udělat si pořádek v základních pojmech a konceptech

Strukturované vs. objektové modelování softwaru

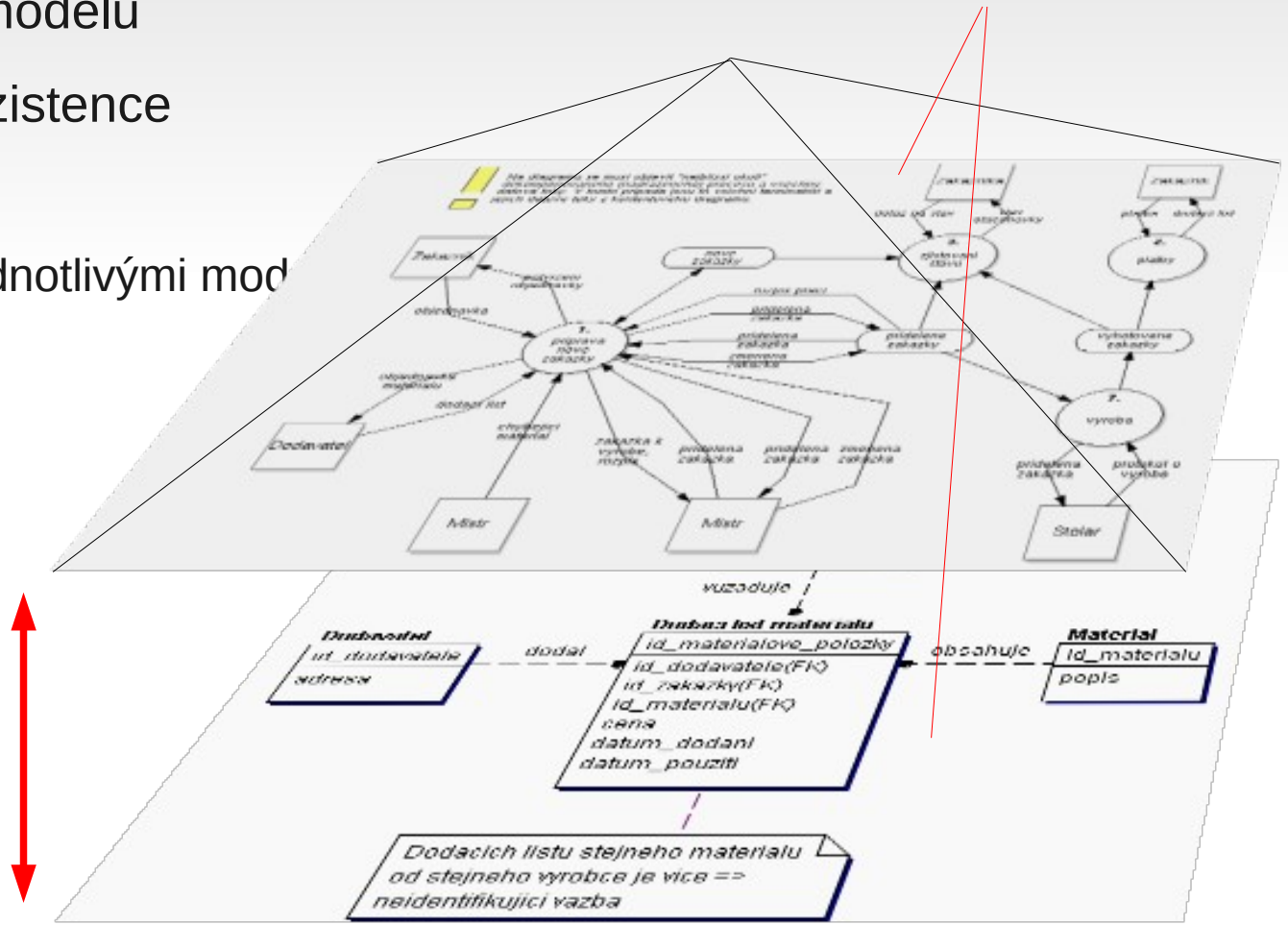
Proč modelovat IS?

- 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“



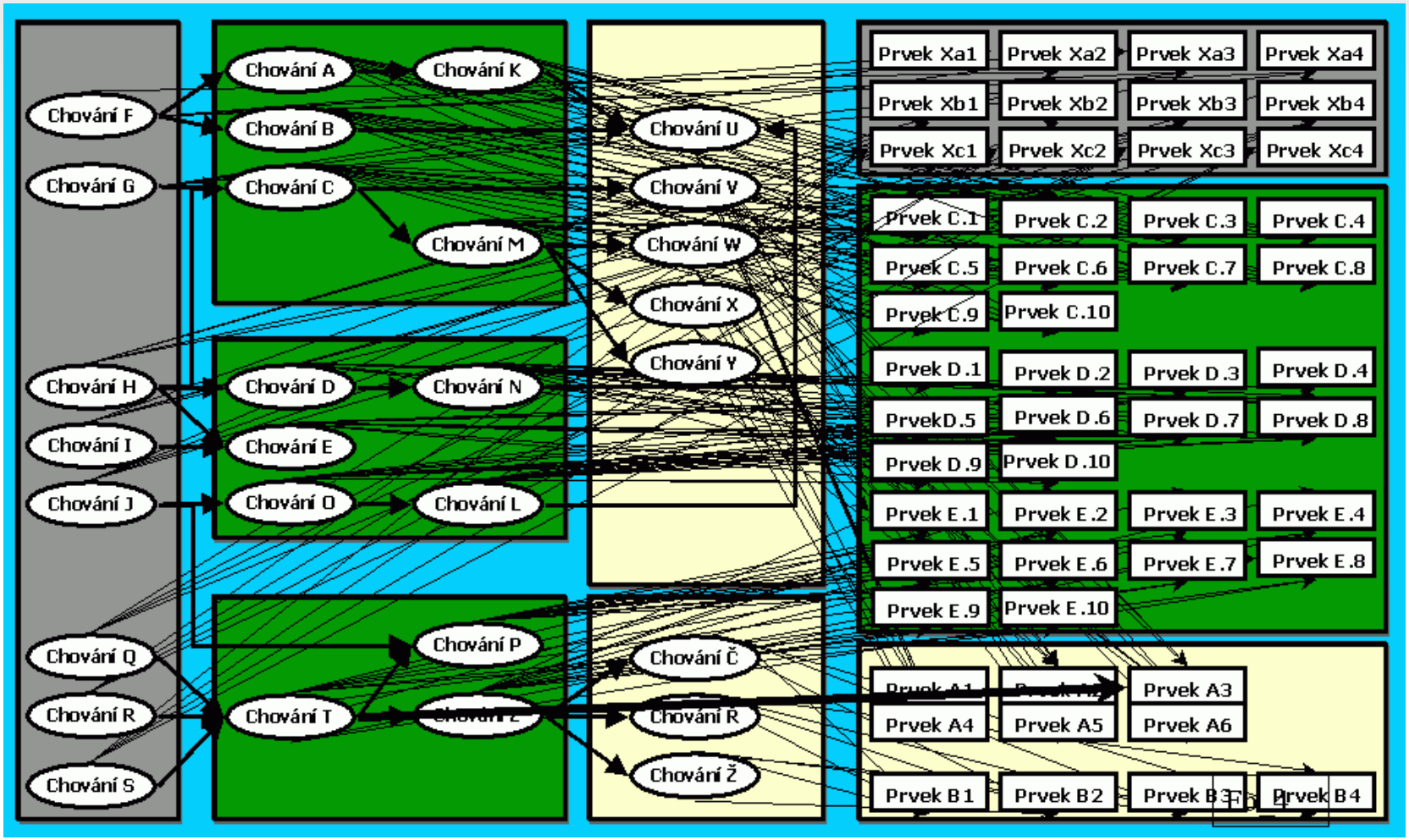
Strukturované modelování

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



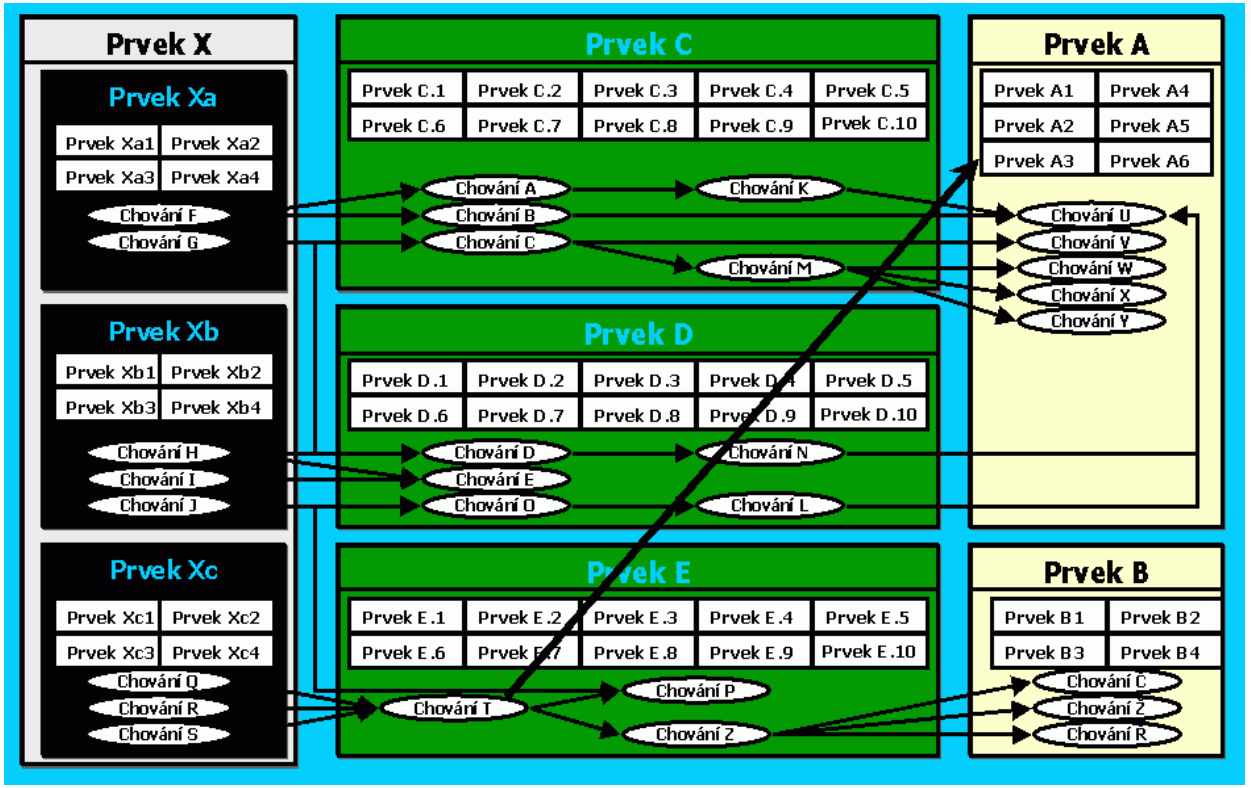
Strukturované modelování pokr.

- 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



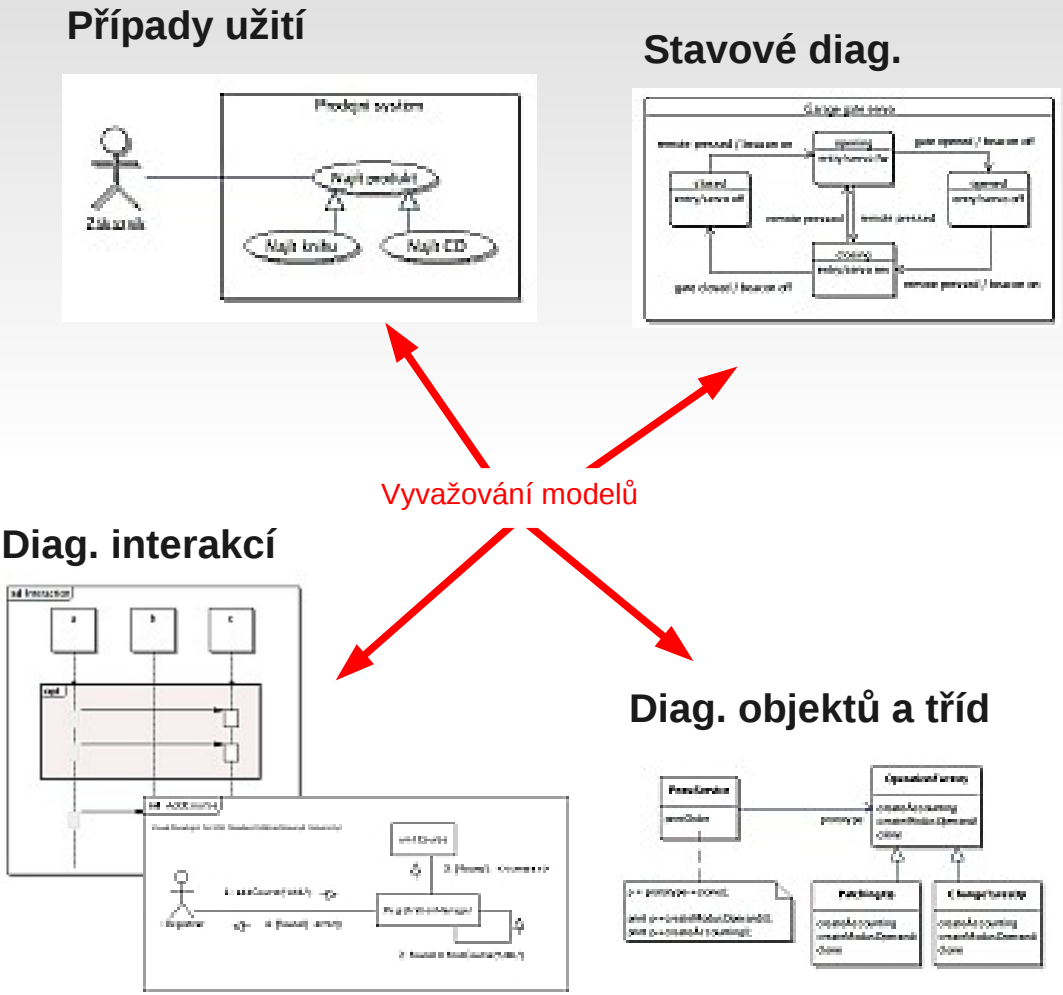
Objektové modelování

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



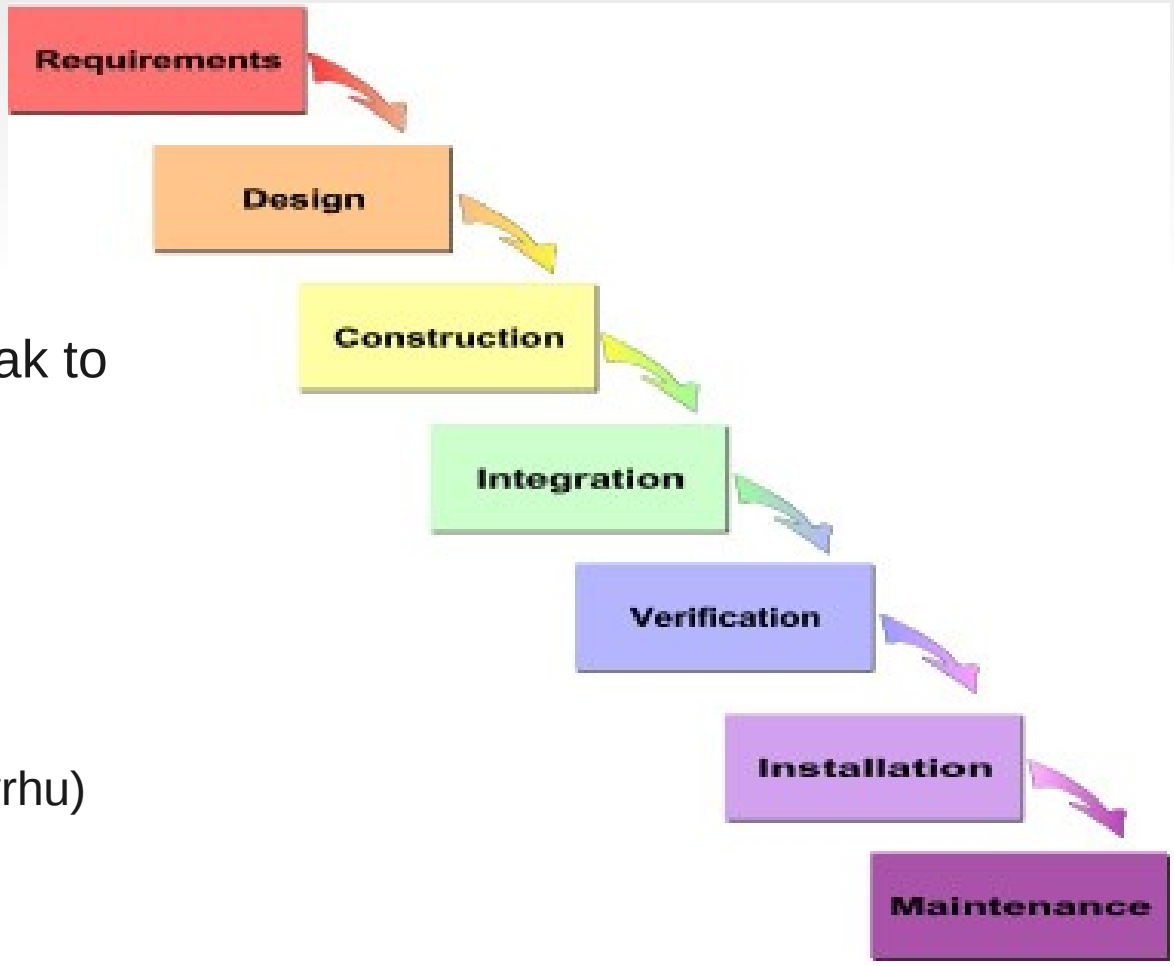
Životní cyklus softwaru
vodopád, iterativní a inkrementální postupy

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 objektovém světě se masivně v praxi 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
  - Překrývaní etap (např. se implementuje ještě během návrhu)
  - Pozdní odhalení chyb
  - Odhad ceny



Vodopád – analogie s rodinným domem

- Kompletní výstavba rodinného domu se zahradou a garáží „na klíč“
- Etapy vývoje:
  - Architekt kompletně navrhne dům, garáž, zahradu, ...
  - Projektant zhotoví všechny technické nákresy
  - Stavební firma vše postaví
  - Rodina se nastěhuje a bydlí
- Problém 1 (návrat do předchozí etapy)
  - Rodina po roce bydlení zjistí, že z krbu opadává omítka
  - Návrat na předchozí etapu => firma opraví krb
  - Relativně levná oprava

- Problém 2 (návrat o několik etap zpět)
  - Rodina po nastěhování zjistí, že krb je sice pěkný, ale nehřeje a kouří dovnitř místnosti. Chyba je v samotném typu a umístění krbu a kouřovodů v domě.
  - Architekt navrhne nové řešení, projektant vyprojektuje, ...
  - Hodně drahá oprava.

Iterativní vývoj

- Iterativně = **„předělávat“**
- Celý projekt se vyvíjí v několika iteracích
- Iterace směřují k postupnému vylepšení, zpřesnění, dodělání nebo opravení části systému
- Každá iterace obsahuje analýzu, návrh, testování apod. (tj. miniaturní vodopád), ale s různou intenzitou, např.:
  - V první iterace 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 testovaní se provádí pouze jednou, výsledkem je hotový systém

Iterativní vývoj – analogie s domem

- Postupné přetváření celé stavební parcely až do finální podoby
- První iterace:
  - Architekt rozvrhne kompletní infrastrukturu (rozvody elektřiny do domu, do garáže i k bazénu, přívod vody do domu a k bazénu apod.). Podrobně rozpracuje samotný dům.
  - Stavební firma postaví samotný dům, ale zatím bez vnějších omítek a dalších „detailů“.
  - Rodina se nastěhuje a (provizorně) bydlí.
- Druhá iterace:
  - Architekt rozpracuje garáž a bazén.
  - Stavební firma dodělá vnější omítky a postaví garáž.
- Třetí iterace:
  - Stavební firma vybuduje bazén a upraví okolí.

=> rychlejší (dílčí) výsledky – rodina dříve bydlí, byť provizorně

=> rychlejší odhalení chyb – rodina zjistí mnohem dříve, že krb je špatný

Inkrementální vývoj

- Inkrementálně = **„přidávat k“**
- Uplatňuje se zejména u větších projektů a/nebo v agilním vývoji
- Jednotlivé části systému (přírůstky, inkrementy) vytváříme „nezávisle“ na zbytku a pak integrujeme
- Vývoj jednotlivých přírůstků může probíhat iterativně, vodopádem, XP, …
  - Nejčastěji se používá iterativní vývoj přírůstků
- Vývoj typu vodopád je tedy inkrementální vývoj s jediným přírůstkem představujícím celý systém

Inkrementální vývoj – analogie s domem

- Postupné přidávání nových věcí k již hotovým
- První přírůstek – rodinný dům:
  - V rámci prvního přírůstku vzniká nový rodinný dům např. metodou iterativního vývoje
  - Výstupem je kompletně hotový a zabydlený dům

- Druhý přírůstek – garáž:
  - V rámci druhého přírůstku vzniká garáž např. pomocí vodopádu (architekt navrhne garáž a její umístění na pozemku, projektant zhotoví technické nákresy garáže, stavební firma garáž postaví)
  - Pro garáž je nutné probourat jednu zeď rodinného domu a upravit elektrorozvodnou skříň, aby šlo přivést elektřinu do garáže => integrace
  - Výstupem je kompletní funkční celek *„dům s garáží“*
- Třetí přírůstek – bazén:
  - Majitel domu si sám navrhne a vybuduje na zahradě bazén
  - Je třeba přivést z domu vodu a elektřinu pro osvětlení bazénu => integrace

=> 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. automatická 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. Model-driven vývoj

Agilní metody vývoje

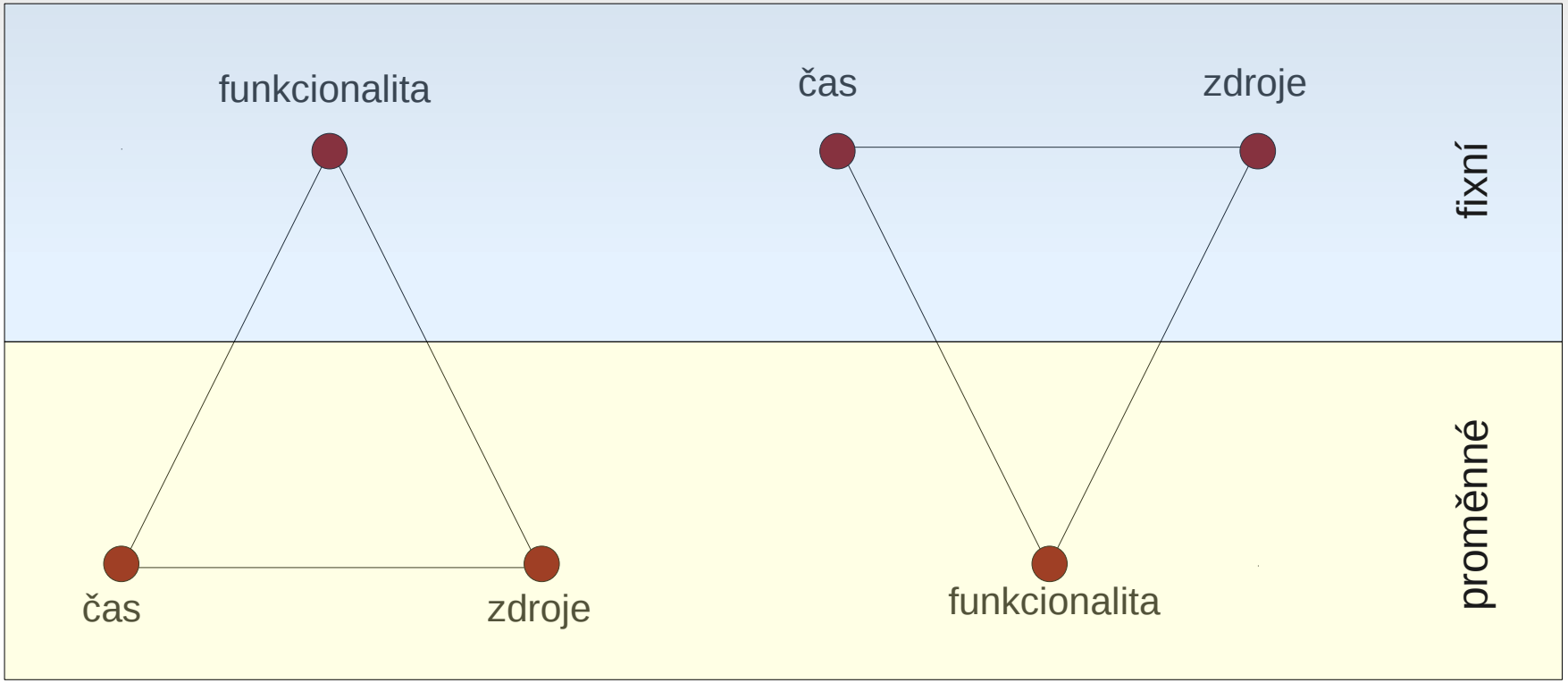
- *Agile development*
- Metody zaměřené více na lidi – určujícím faktorem v úspěchu projektu je kvalita lidí pracujících na projektu a jejich spolupráce
- Velmi krátké iterace (jeden měsíc a méně)
- **UML se používá pouze jako doplněk**
- Malé ale výkonné týmy (dvojice)
- Zapojení zákazníka do vývoje (zákazník se zúčastní 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

Vývoj řízený modely

- *Model-Driven Engineering, MDE*
- „Seriózní“ přístup, kdy implementujeme striktně na základě UML modelů
  - Průběžně vniká 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 funkcionalita, dostupné zdroje a čas



model-driven engineering

agilní přístup

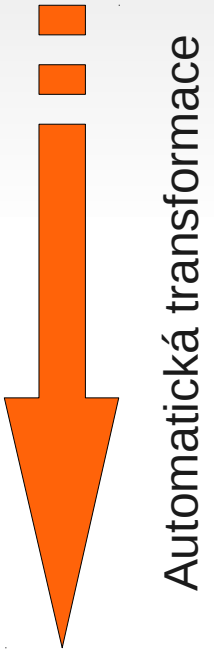
Softwarové architektury – MDA

Softwarové architektury

- Architektonické modely a metody vývoje
- Např.:
  - model-driven architecture – MDA
  - component architecture
  - service-oriented architecture – SOA
  - …
- Detaily budou probrány ve specializovaných přednáškách ke konci semestru

Model-Driven Architecture - MDA

- OMG standard definující rozsah softwarových modelů, způsob jejich vytvoření a použití
- Rozvíjí myšlenku vývoje pomocí modelů (MDE) o *formalizaci* modelů a jejich *automatickou transformaci* (automatizované zpřesňování modelů)
- MDA definuje 4 úrovně modelů:
  - CIM – Computation Independent Model
    - Model nezávislý na počítačovém zpracování (business analýza)
  - PIM – Platform Independent Model
    - Platformově nezávislý model řešení
  - PSM – Platform Specific Model
    - Platformově specifický model řešení
  - Code
    - Kód aplikace, tj. výsledná realizace řešení
- **MDA definuje způsob (automatické) transformace modelů**



MDA: Computation Independent Model

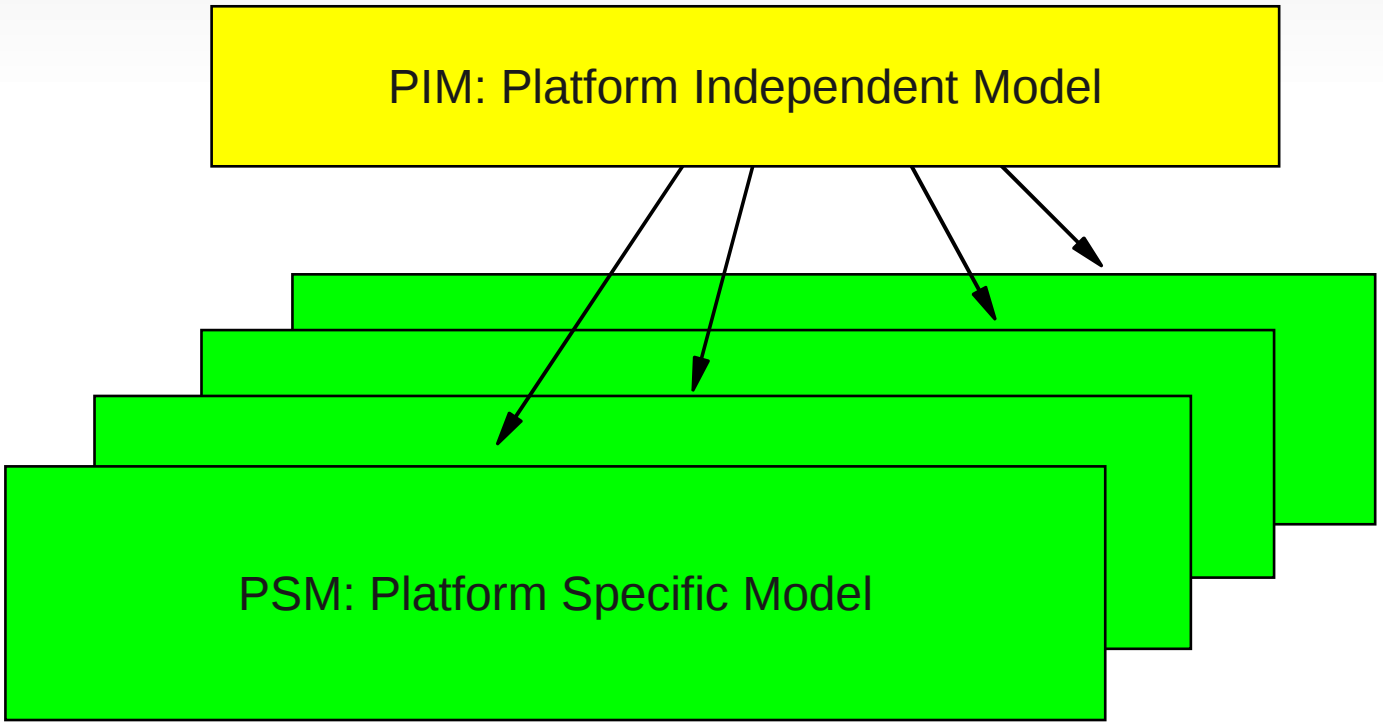
- Business analýza = model procesních toků uvnitř firmy
  - Model podnikových procesů
    - V UML nemá speciální diagram, dá se ale použít diagram aktivit
  - Slovník pojmů problémové oblasti
    - U složitých oblastí se dá vyjádřit pomocí konceptuálního diagramu tříd
- CIM vytvářejí buďto sami uživatelé nebo business analytici
- Zaceluje mezeru mezi experty v modelované problematice a experty na návrh a implementaci systémů

MDA: Platform Independent Model

- Analýza, business architektura
- Koncepční řešení problematiky na základě konkrétních požadavků
- Neobsahuje informace spojené s konkrétní technologií realizace
- PIM vytvářejí IT analytici

MDA: Platform Specific Model

- Návrh, technologické řešení
- Řešení pro zvolenou platformu (J2EE, CORBA, ...)
- Respektuje technologické standardy a návrhové vzory
- Pro jeden PIM může být několik platformě-specifických modelů
- PSM vytvářejí návrháři



UML

UML and/or
Platform specific notation

Platforms: Web Services, ebXML, J2EE/EJB, CORBA, MS .Net, …

MDA: Kód

- Obsah dodávky
- Kód je synchronizován s návrhovým modelem
- Z hlediska MDA je kód chápán jako model konkrétní realizace na dané platformě

MDA: Transformace PIM -> PSM

- Transformace Analýza -> Návrh
- Postup:
  - PIM je doplněn mapovacími značkami, které definují, jaká obecná transformační pravidla budou použita
    - např. přiřazení obecnějšího návrhového vzoru příslušnému modelovanému elementu
  - Pro PIM model (resp. jeho části) je zvolena implementační platforma
  - Na základě mapovacích značek jsou provedeny odpovídající transformace již s ohledem na zvolenou platformu
    - dávky připravených pravidel pro: Java, C#, C++, .Net, ...
    - automaticky se vytvářejí nové třídy, rozhraní, atributy a operace
    - Na základě mapování tříd do databáze jsou generovány SQL příkazy

Synchronizace PIM <--> PSM

- Obousměrná synchronizace důležitá pro iterativní a inkrementální vývoj
- Změny pouze s implementační charakteristikou nejsou promítány do PIM

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. d*evelopment case*)
  - Často se přizpůsobuje i firmě a nástrojům

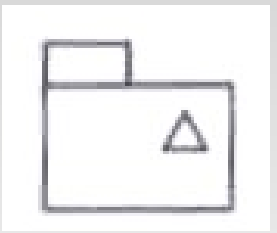
Notace RUP

- **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 jednotlivec nebo tým může vystupovat v několika *workers.*
  - v RUPu se používá termín *role* namísto *worker*, význam je ale stejný

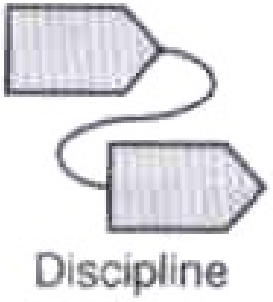
- **Aktivita** *(activity)*
  - modeluje „co“ v procesu vývoje softwaru
  - úkol v projektu vykonávaný jednotlivci nebo týmy
  - jednotlivci a týmy musí při provádění aktivity vždy přijmout konkrétní roli; UP i RUP proto s aktivitami asociuje role (workers)
  - aktivity mohou být dekomponovány na podrobnější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

Notace RUP pokr.

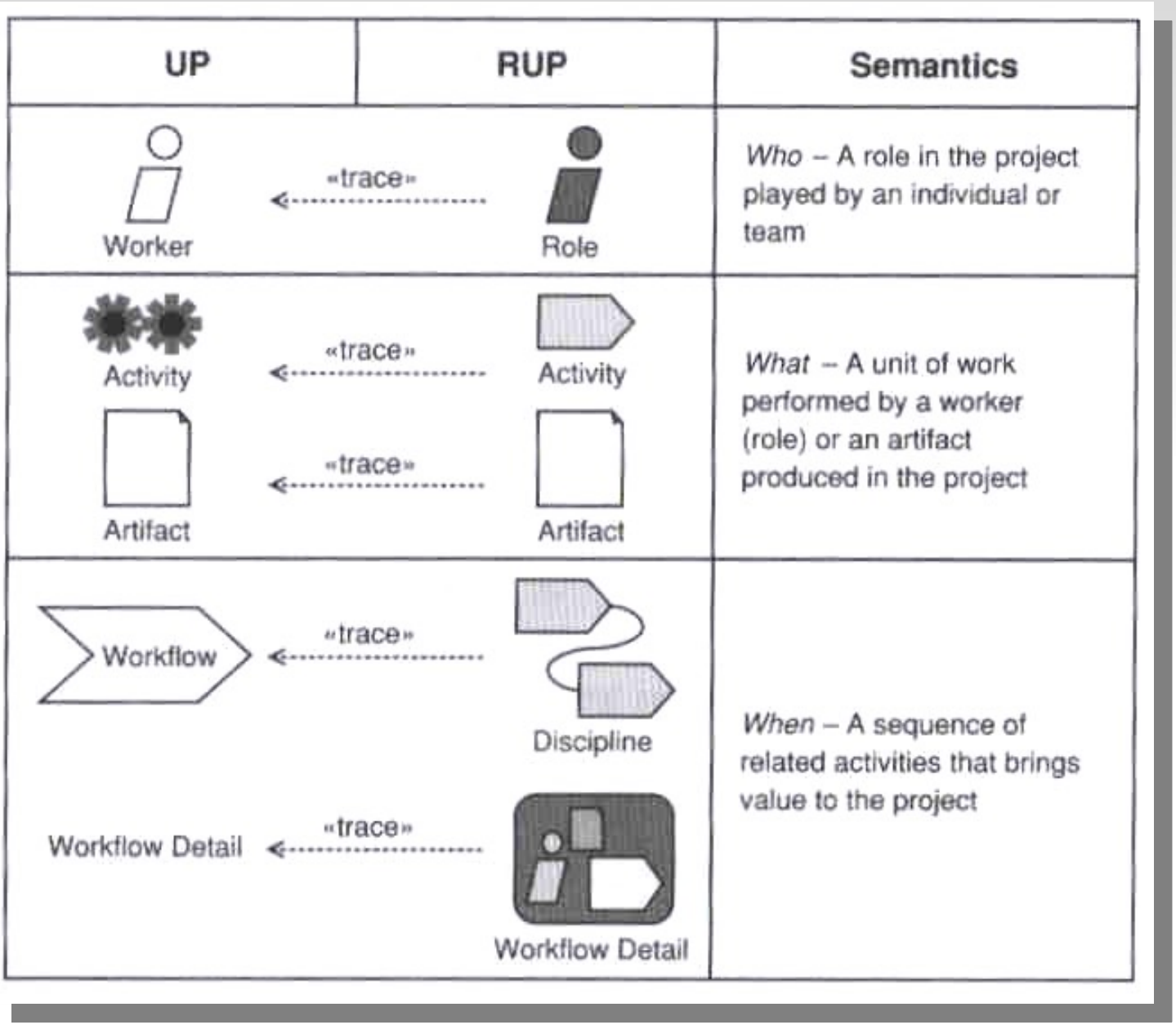
- **Artefakt** *(artifact)*
  - modeluje „co“ v procesu vývoje softwaru
  - vstupy a výstupy projektu
    - zdrojový kód, spustitelný program, standardy, dokumentace, ...
  - mohou být znázorněny různými ikonami v závislosti na tom, co představují



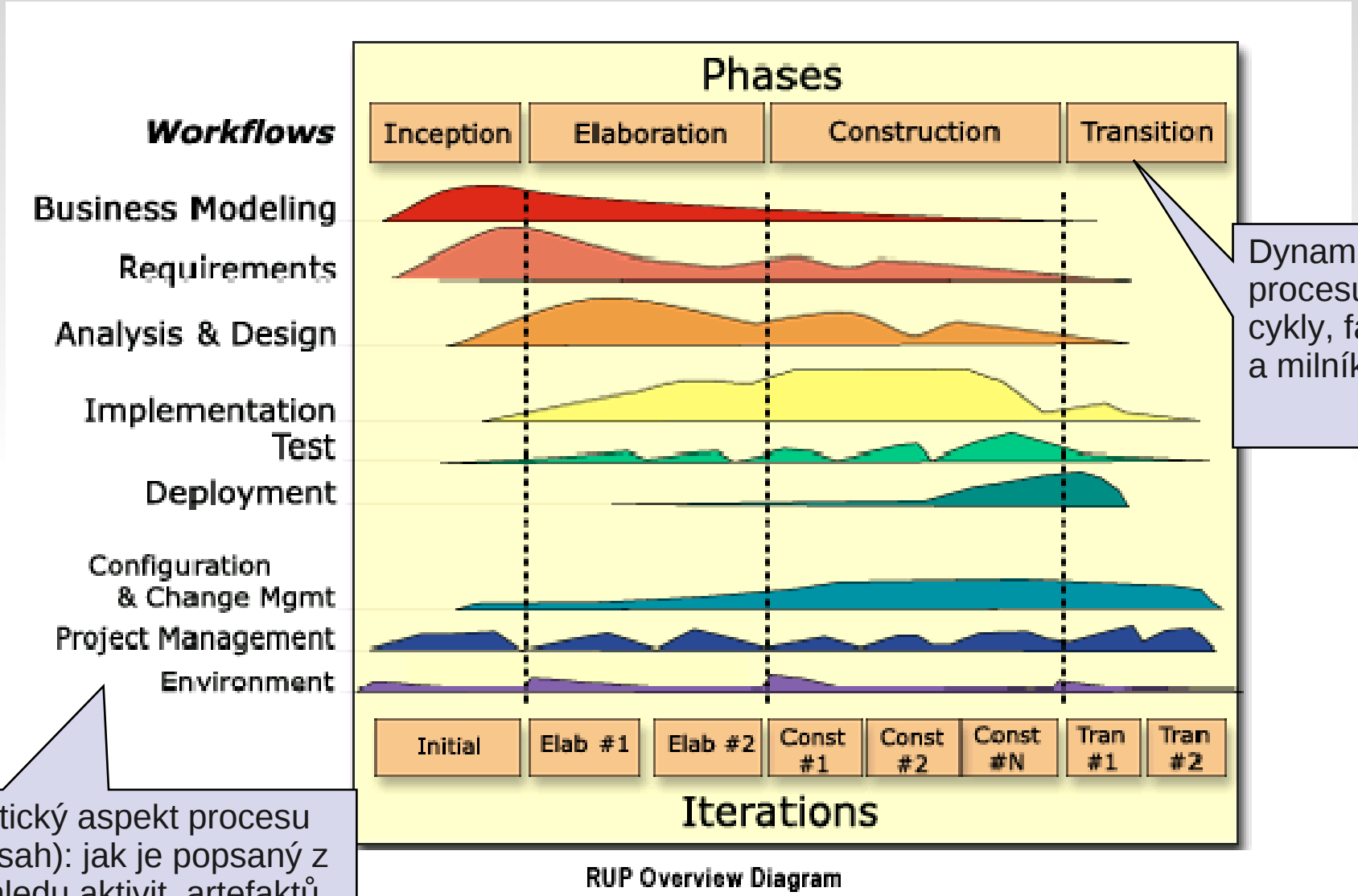
- **Tok práce** *(workflow)*
  - modeluje „kdy“ v procesu vývoje softwaru
  - sekvence aktivit vykonávaných pracovníky (workers)
  - mohou být dekomponovány na jeden a více detailnějších modelů
  - detailnější toky práce jsou pouze odkazovány ikonou



Notace UP vs. RUP



RUP: Dvě dimenze vývoje softwaru

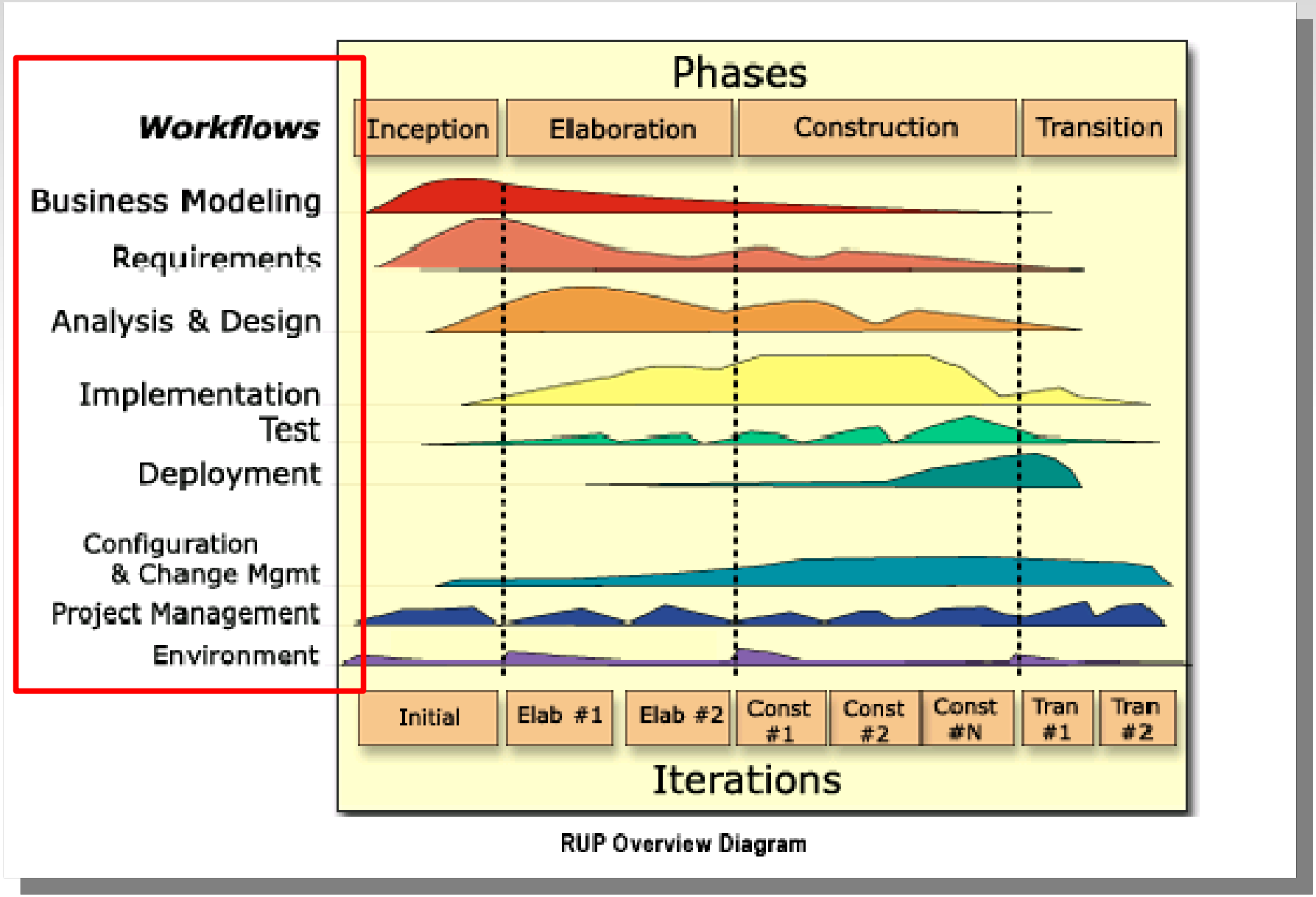


RUP Overview Diagram

Dynamický aspekt procesu (čas): cykly, fáze, iterace a milníky.

Statický aspekt procesu (obsah): jak je popsaný z pohledu aktivit, artefaktů, pracovníků a toků práce.

RUP: Workflows



RUP Overview Diagram

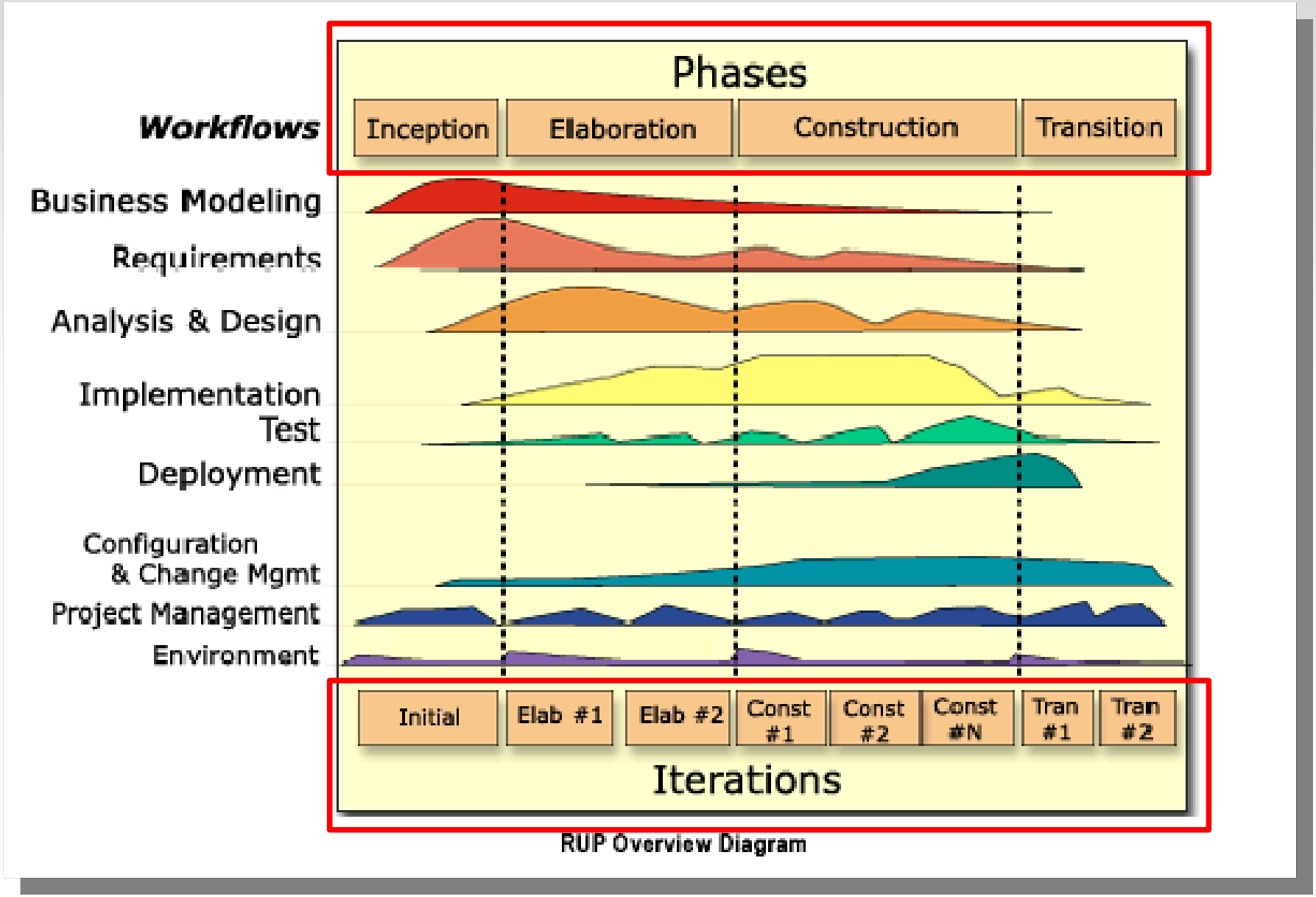
RUP: Workflows (I)

- 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
  - Dokumenatce use-casů, nejprve jednou větou, později scénáře, toky událostí, apod.
- Analysis and Design
  - Nalezení základních objektů, tříd a balíků, 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
  - Testování jednotek (Unit Testing)
  - Integrační testy (Integration Testing)
  - Testování systému (systém 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

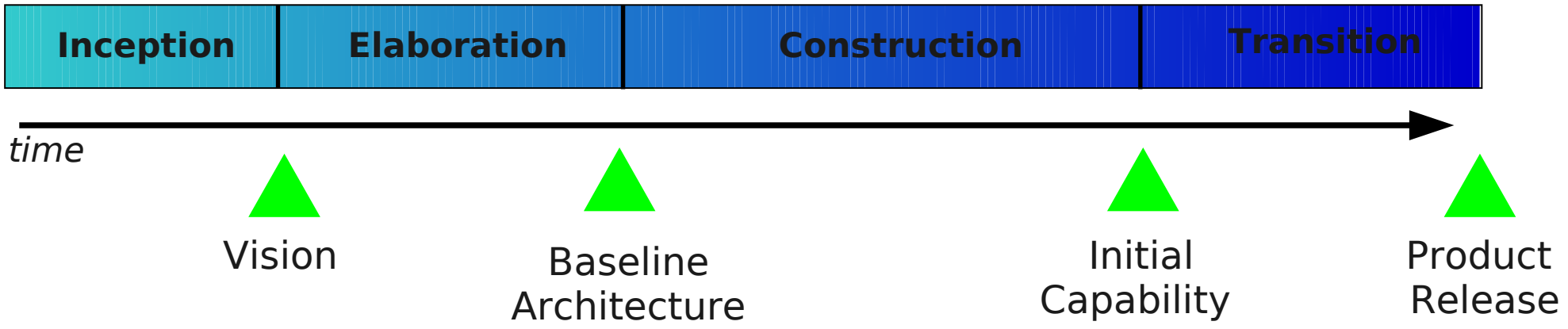


RUP Overview Diagram

RUP: Cykly, fáze, iterace (I)

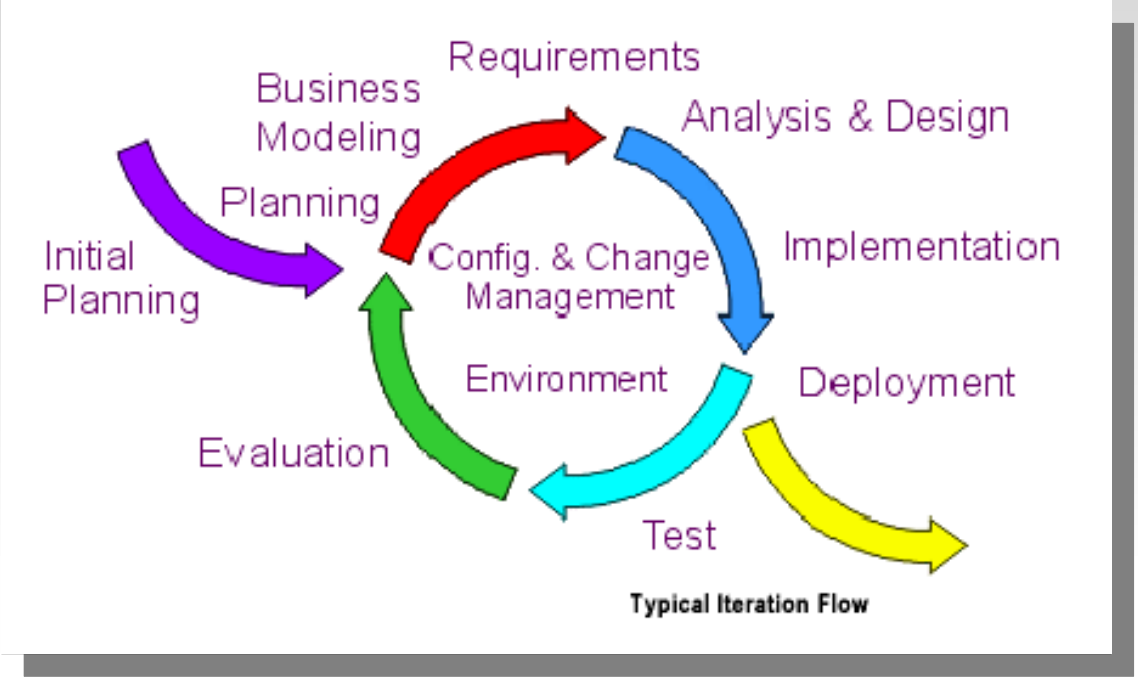
- Výrobek prochází různými **cykly** (= inkrementy v inkrementálním vývoji)
  - každý cyklus produkuje novou verzi systému (release)
- Každý cyklus se skládá z těchto **fází**:

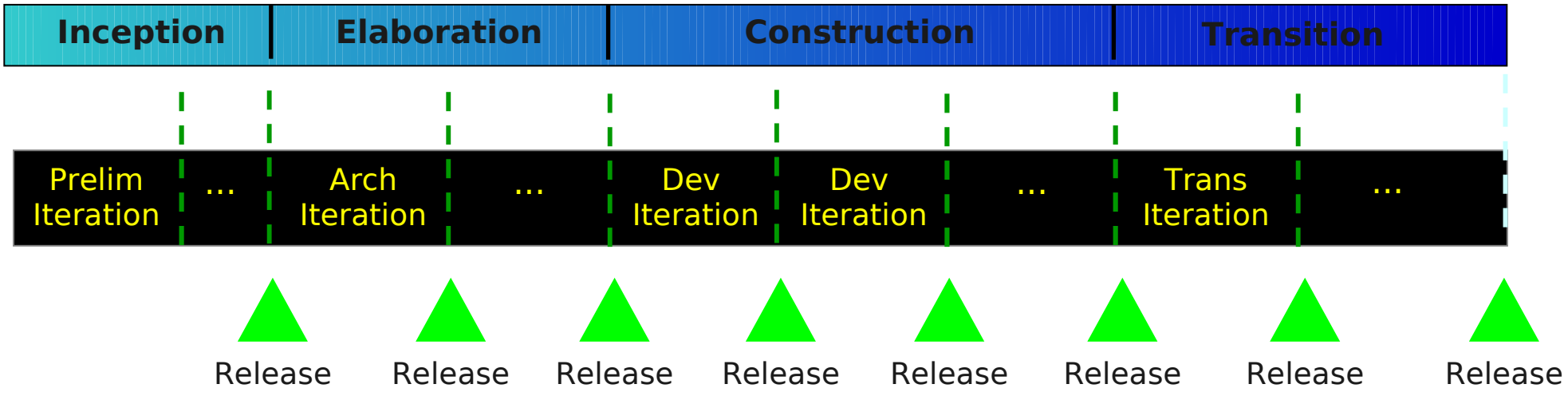
  - **Zahájení (Inception)**: potvrzení konceptu, proveditelnost, cíle
  - **Rozpracování (Elaboration)**: detailní požadavky, scénáře použití, architektura, komponenty, vzory interakce na vysoké úrovni
  - **Konstrukce (Construction)**: zdokonalení architektury, implementační iterace řízené rizikem
  - **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
  - “post-morten” analýza systému
- **Milník**
  - nová verze (product release)