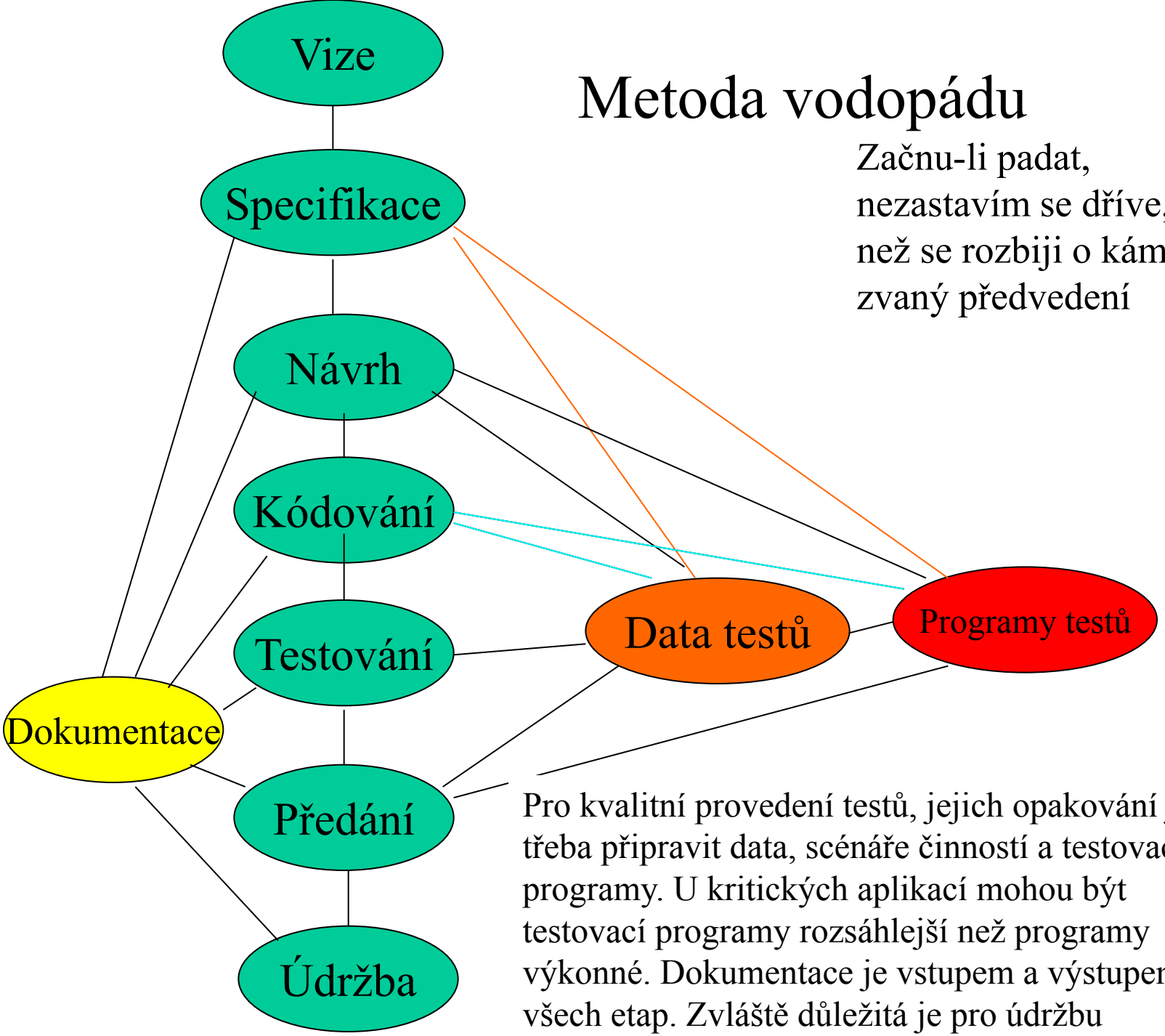
Procesy vývoje softwaru

Coherent sets of activities for specifying, designing, implementing and testing software systems

Promyšlená síť aktivit prováděných při specifikaci požadavků, návrhu, implementaci, testování (a údržbě) softwarových systémů

Metoda vodopádu

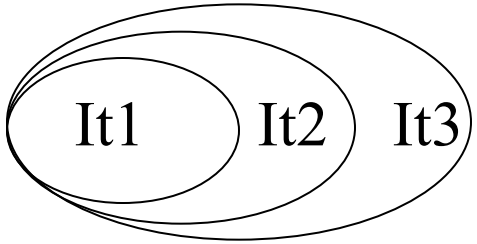
Začnu-li padat,
nezastavím se dříve,
než se rozbiji o kámen
zvaný předvedení



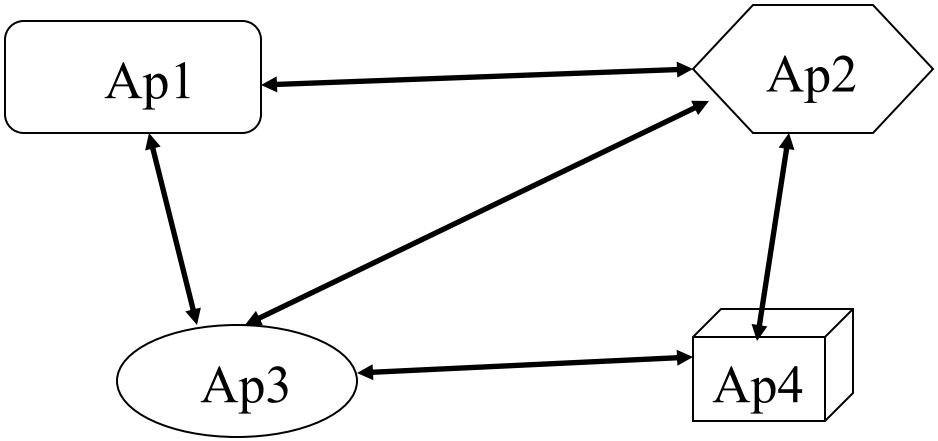
Pro kvalitní provedení testů, jejich opakování je třeba připravit data, scénáře činností a testovací programy. U kritických aplikací mohou být testovací programy rozsáhlejší než programy výkonné. Dokumentace je vstupem a výstupem všech etap. Zvláště důležitá je pro údržbu

Vodopád je jen někdy užitečný, je tedy nutné ho modifikovat

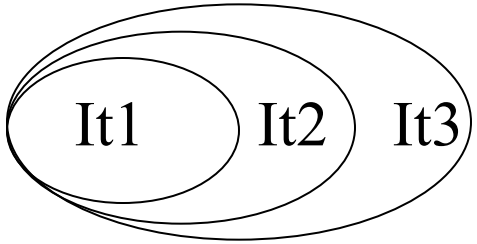
1. Každá etapa (často i některé její kroky) životního cyklu se podrobí různým formám kontroly (čtení kódu, inspekce, …)
2. Specifikace se otestují pomocí prototypových řešení. Softwarový prototyp je model řešení realizující nebo simulující některé vlastnosti projektovaného systému. Tím se případné nedostatky odhalí již v etapě specifikace požadavků
3. Projekt se realizuje postupně rozšiřováním jistého jádra o relativně samostatně vyvíjené přírůstky – samostatné aplikace (inkrementální realizace) nebo doplňováním funkcí do dané aplikace (iterativní vývoj).



Iterativní vývoj –jedna rostoucí aplikace (může být distribuovaná), pro vývoj iterace potřebuji mít již vyvinuté jádro a **to potřebuji při vývoji iterace používat. Typické pro agilní vývoj**

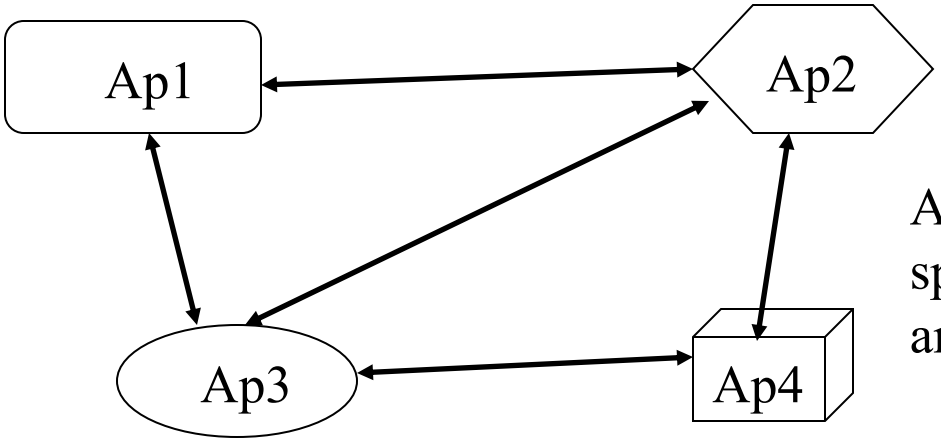


Inkrementální vývoj, propojují se (různorodé) aplikace Ap1, Ap2, .. většinou pomocí výměny zpráv nebo dat přes middleware, nebo přístupu ke společným datům (datovým úložištím). Při vývoji přírůstku **nepotřebuji mít k dispozici zbytek systému!!!, Agilita potřebuje doladit**



Obvyklé rozhraní: RPC, API

Iterativní vývoj –jedna rostoucí aplikace (může být distribuovaná), pro vývoj iterace potřebuji mít již vyvinuté jádro a **to potřebuji při vývoji iterace používat. Typické pro agilní vývoj**



Asynchronní komunikace
společná DB (menší systémy)
architekturní služby

Inkrementální vývoj, propojují se (různorodé) aplikace Ap1, Ap2, .. většinou pomocí výměny zpráv nebo dat přes middleware, nebo přístupu ke společným datům (datovým úložištím). Při vývoji přírůstku **nepotřebuji mít k dispozici zbytek systému!!!, Agilní vývoj potřebuje specifické nástroje**

Závislost procesu na typu řešení

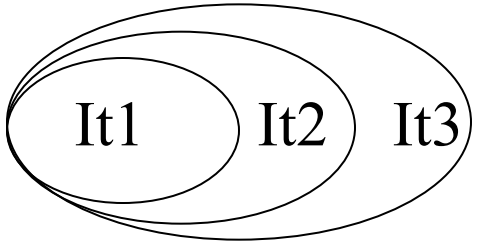
- Skoro vodopád:
  - Kritická řízení technologií, kdy lze jen přípustit nedokonalé specifikace
  - Specifikace musí být téměř OK, to je potřeba při přijímání norem
- Modifikovaný vodopád:
  - Vývoj SW pro hromadné použití
    - Hry, OS, základní aplikace
      - Velké firmy nebo open source

Modifikovaný vodopád pro hromadný SW

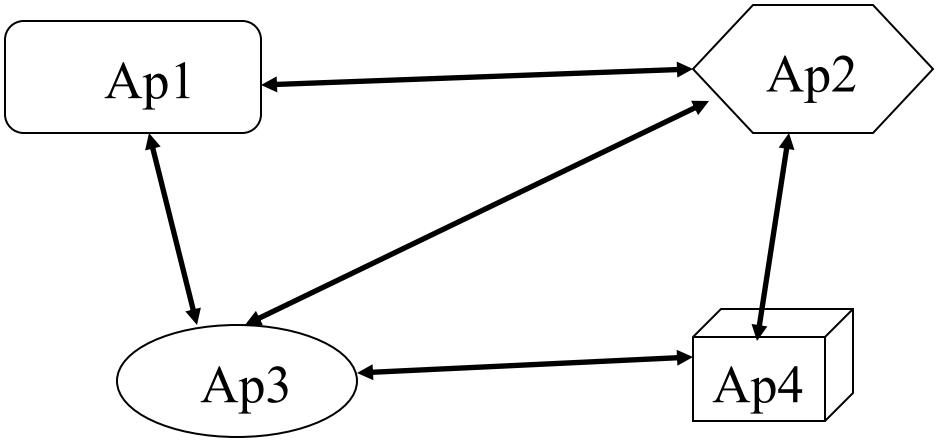
1. Pečlivé specifikace +oponentury uživatelů
2. Návrh
3. Kódování
4. Alfa testování ve vývojovém týmu,
5. Beta testování vybranými uživateli, testuje se celková vlastnosti,
   - Vlastně pokročilé prototypování
6. Distribuce (předání),
7. Permanentní úpravy, údržba pro mnoho uživatelů současně
8. Lze realizovat i inkrementální vývoj v bodech 1. až 4., používá se ale zřídka

Modifikovaný vodopád při customizaci

1. Vize
2. Specifikace +oponentury uživatelů
3. Návrh, co se bude a jak customizovat, návrh pro nově vyvíjené části
4. Kustomizace, doplňkové kódování
5. Oživování a testování
6. Předání a podpora provozu
7. Lze realizovat i inkrementální vývoj, málo se podporuje, trochu se to ale mění



Iterativní vývoj –jedna rostoucí aplikace (může být distribuovaná), pro vývoj iterace potřebuji mít již vyvinuté jádro a **to potřebuji při vývoji iterace používat**



Inkrementální vývoj, propojují se (různorodé) aplikace Ap1, Ap2, .. většinou pomocí výměny zpráv nebo dat přes middleware, nebo přístupu ke společným datům (datovým úložištím). Při vývoji přírůstku **nepotřebuji mít k dispozici zbytek systému!!!**

Hlavní důvody tvorby prototypů

  - Softwarový prototyp se jako částečně funkční model cílového řešení vytváří z následujících důvodů:
    - ověření specifikací funkcí (úplnost, správnost) a návrhu (struktury systému),
    - předběžný odhad nákladů a rizik realizace.
- S možnou výjimkou SOA se prototyp nakonec zahodí a vše se napíše znovu. V SOA se dá používat i během provozu jako prostředek simulace neexistujících částí, ruční provoz a ladění (přesměrování zpráv)
  - Zahazování prototypů se vřele doporučuje v literatuře, nejsem si ale jist, zda to vždy platí. Důvodem je nebezpečí, že se převezme řešení, které je nedotažené
  - Např. pro prototypy používané v SOA (simulátory UI)
- Variantou prototypu, je nástřel – ověření, že něco může fungovat. Osvědčené řešení se použije/integruje

Typy prototypů

a) *Potěmkin* (Obrazovkový prototyp, mock up prototype): Model cílového systému, který simuluje obrazovky dialogů. Vlastní výkonná část systému zcela (nebo téměř zcela) chybí. Tento typ prototypu vlastně simuluje budoucí rozhraní systému.

b) *Jiný kůň:* Systém je téměř úplný, ale funguje na jiném hardwaru resp. nad jiným základním softwarem. Časté pro software pro jednočipové počítače.

Typy prototypů

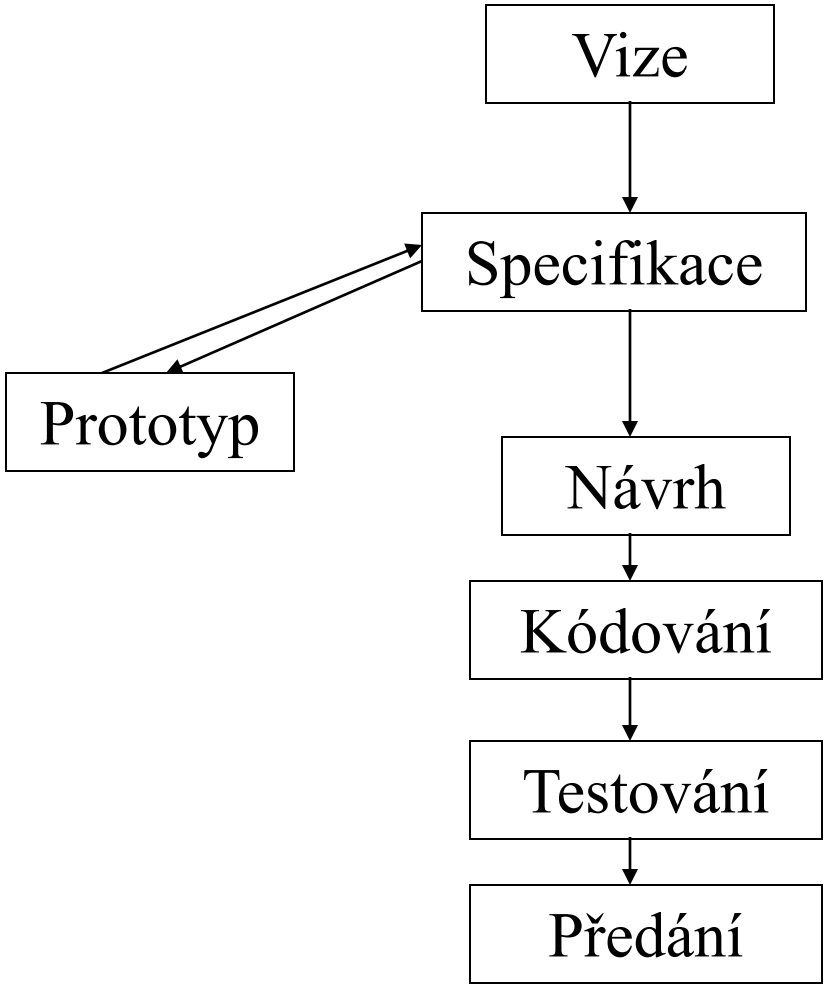
c) *Hlemýžď*: Prototyp je realizován v jazyce, který neumožňuje cílovou efektivnost (např. v jazyce PROLOG)

d) *Nerudný:* Prototyp není uživatelsky příjemný, nemusí být ani dostatečně stabilní.

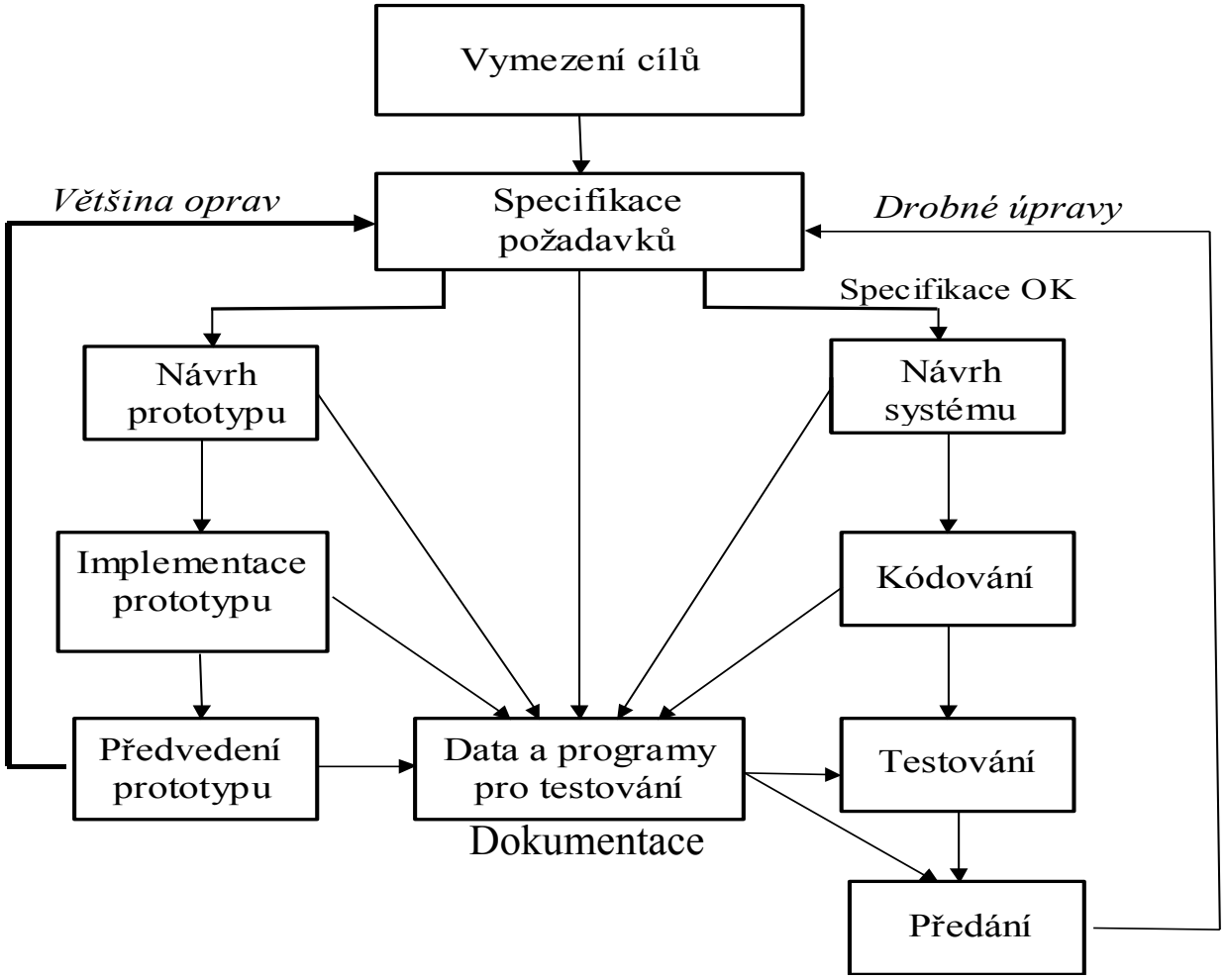
e) *Záludný:* Nereaguje správně na chyby v datech a chyby obsluhy.

f) *Samotář.* Není schopen propojování (PROLOG)

Použití prototypu, klasická varianta



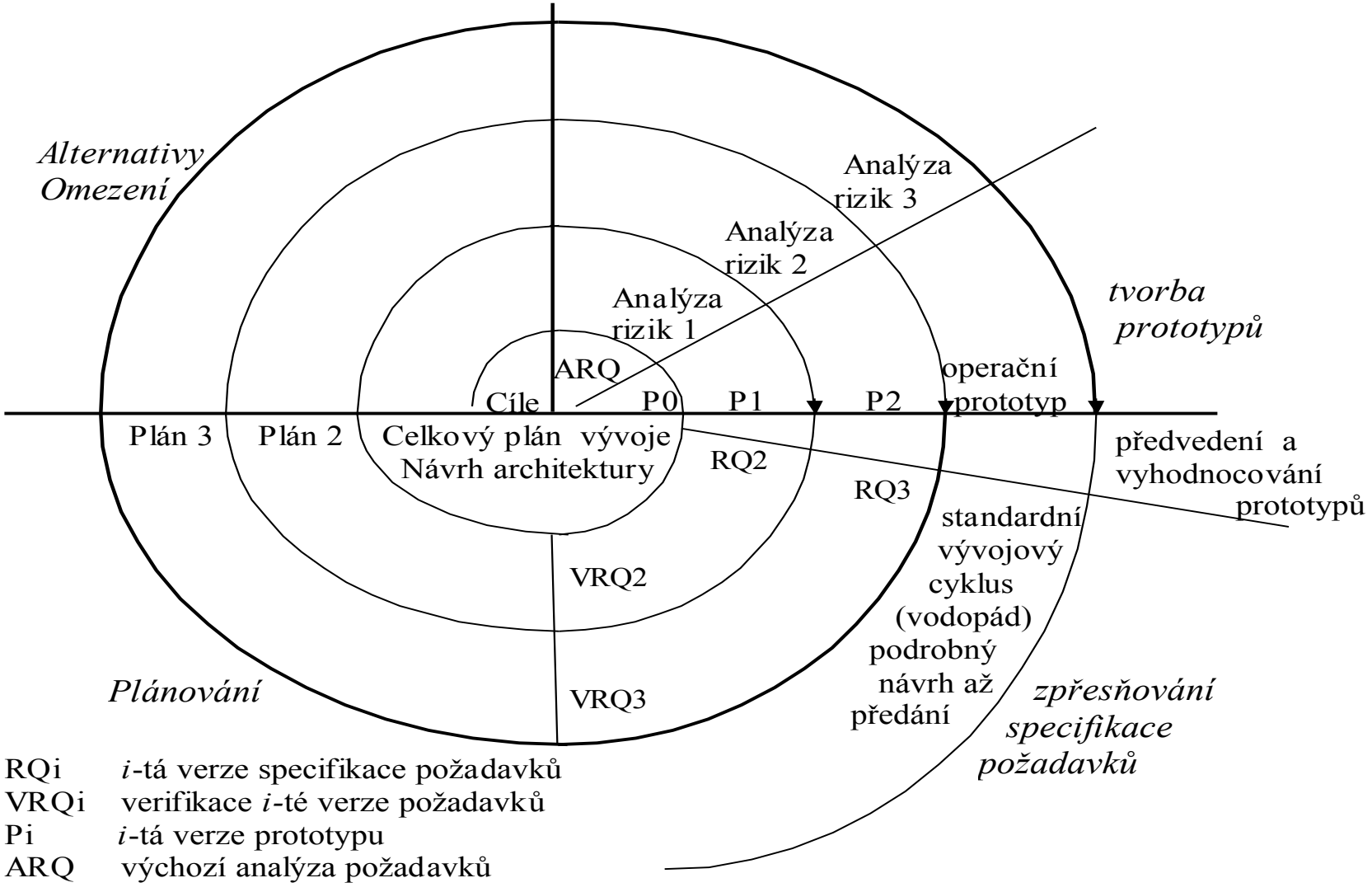
Použití prototypu, promyšlenější použití výsledků



Spirálový model

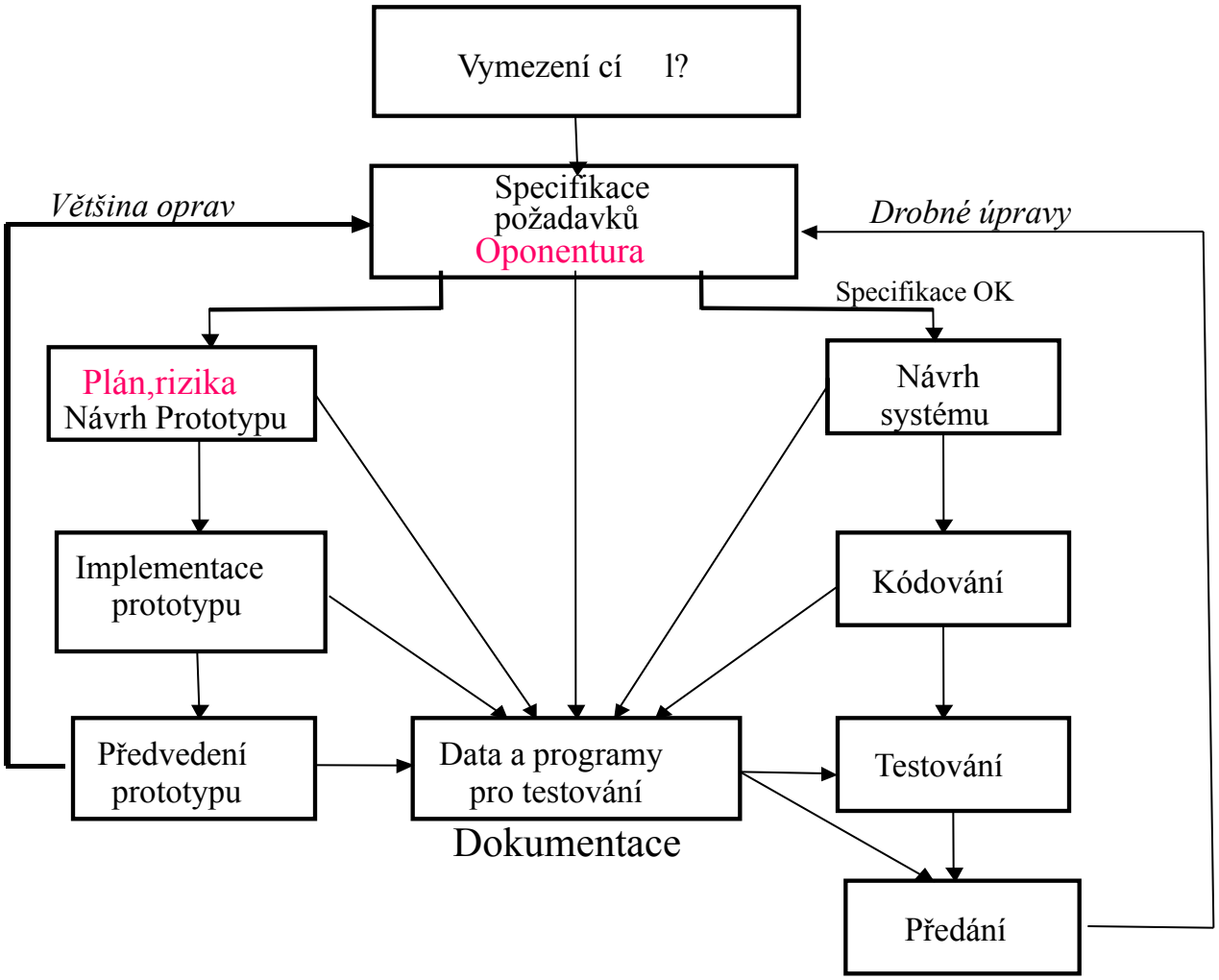
- Stanoví se cíle a výchozí analýza požadavků a návrh architektury
  1. Výchozí prototyp a jeho předvedení
  2. Plán vývoje prototypu
  3. Provede se analýza alternativ a rizik
- Několikanásobně se provede životní cyklus prototypu
  a) Vytvoří zpřesněný prototyp a předvede se
  b) Podle něho se upraví požadavky
  c) Verifikují se (oponují)
  d) Plán vývoje prototypu
  e) Provede se analýza alternativ a rizik
  f) Činnost se opakuje od a)
- Poslední prototyp se nazývá operační. Po něm následuje zbytek vodopádu (návrh, kódování, testování, předání, údržba)

Spirálový model



RQi *i*-tá verze specifikace požadavků
VRQi verifikace *i*-té verze požadavků
Pi *i*-tá verze prototypu
ARQ výchozí analýza požadavků

Použití prototypu

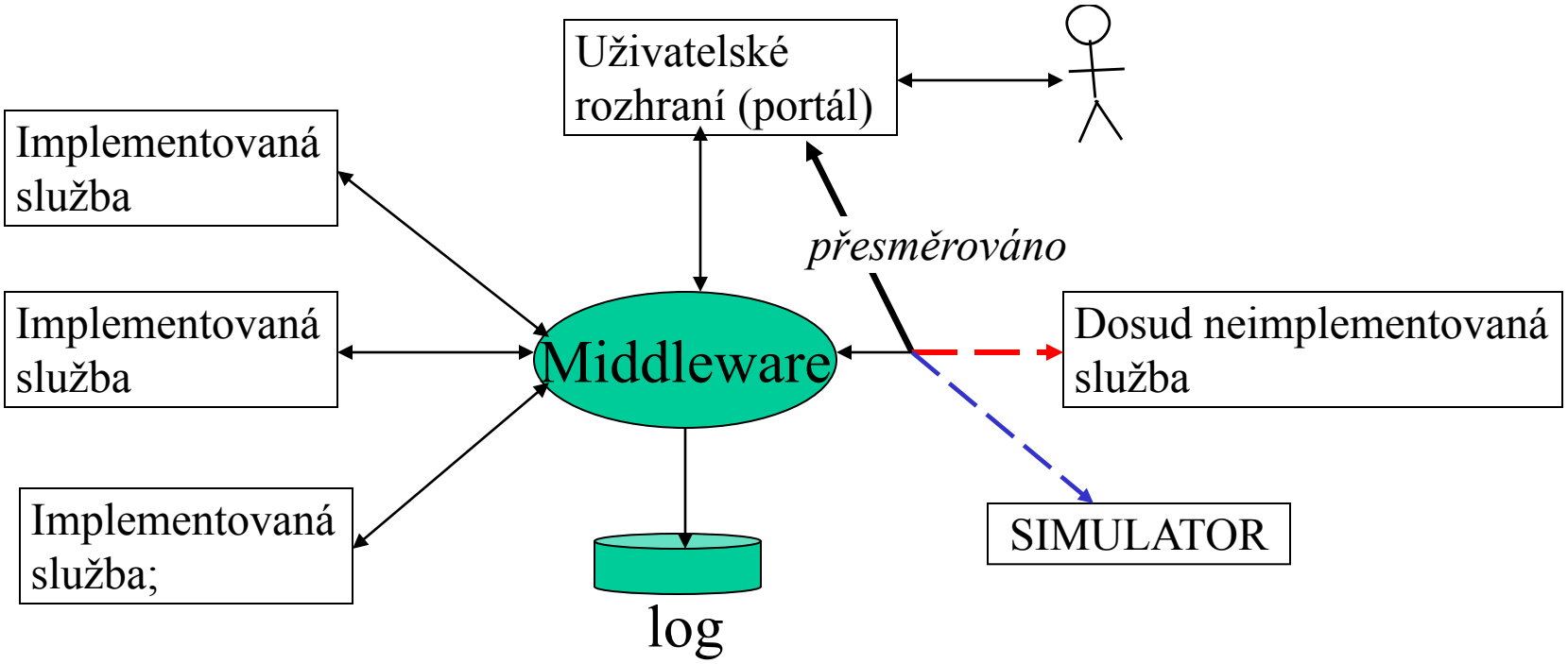


Pozor! V OO se prototyp vyhazuje, v SOA se dá obrazovkový prototyp používat stále, je to i nástřel

Prototypování v SOA

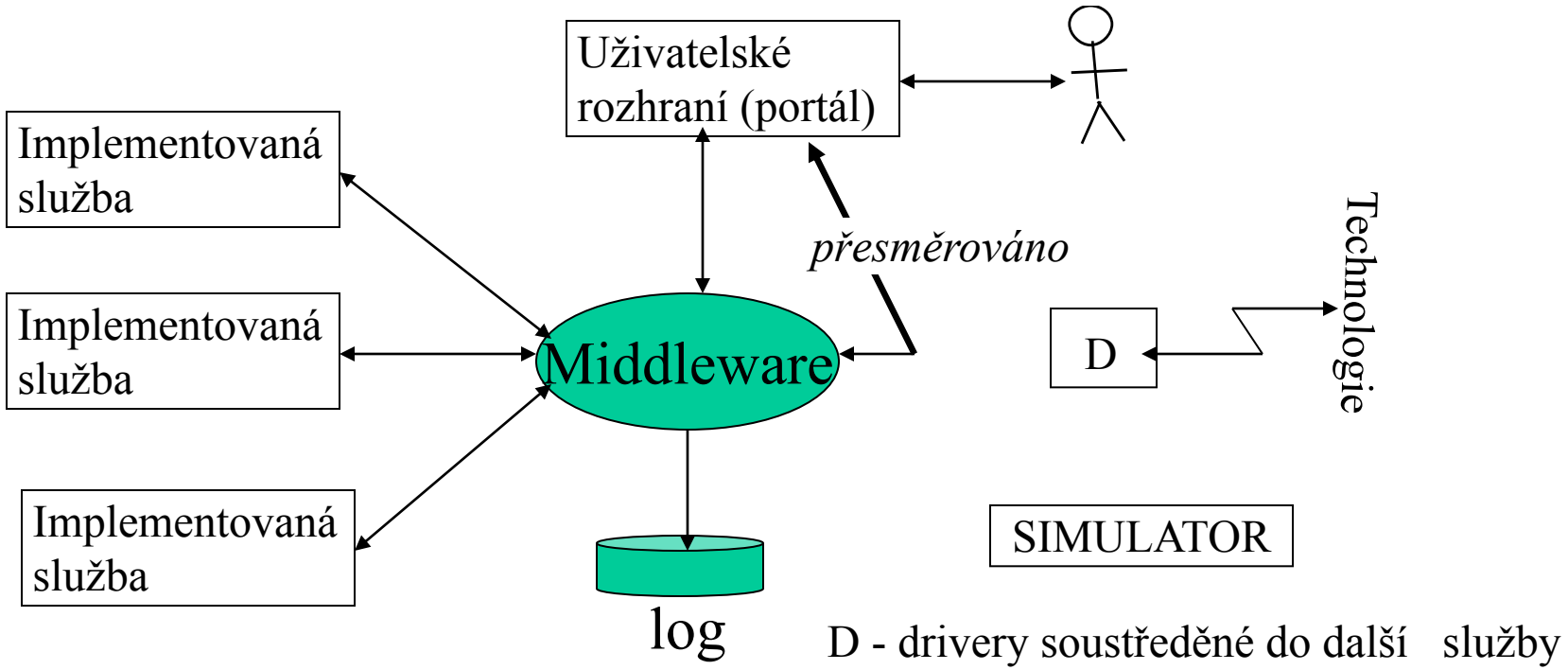
Opakování

Techniky 2
Prototypování, ladění RT systémů



Zprávy lze přesměrovat **beze změny implementovaných** služeb buď na uživatelské rozhraní (efektivní prototyp, použitelné i za běhu systému), nebo dokonce na simulátor (ladění RT systémů)

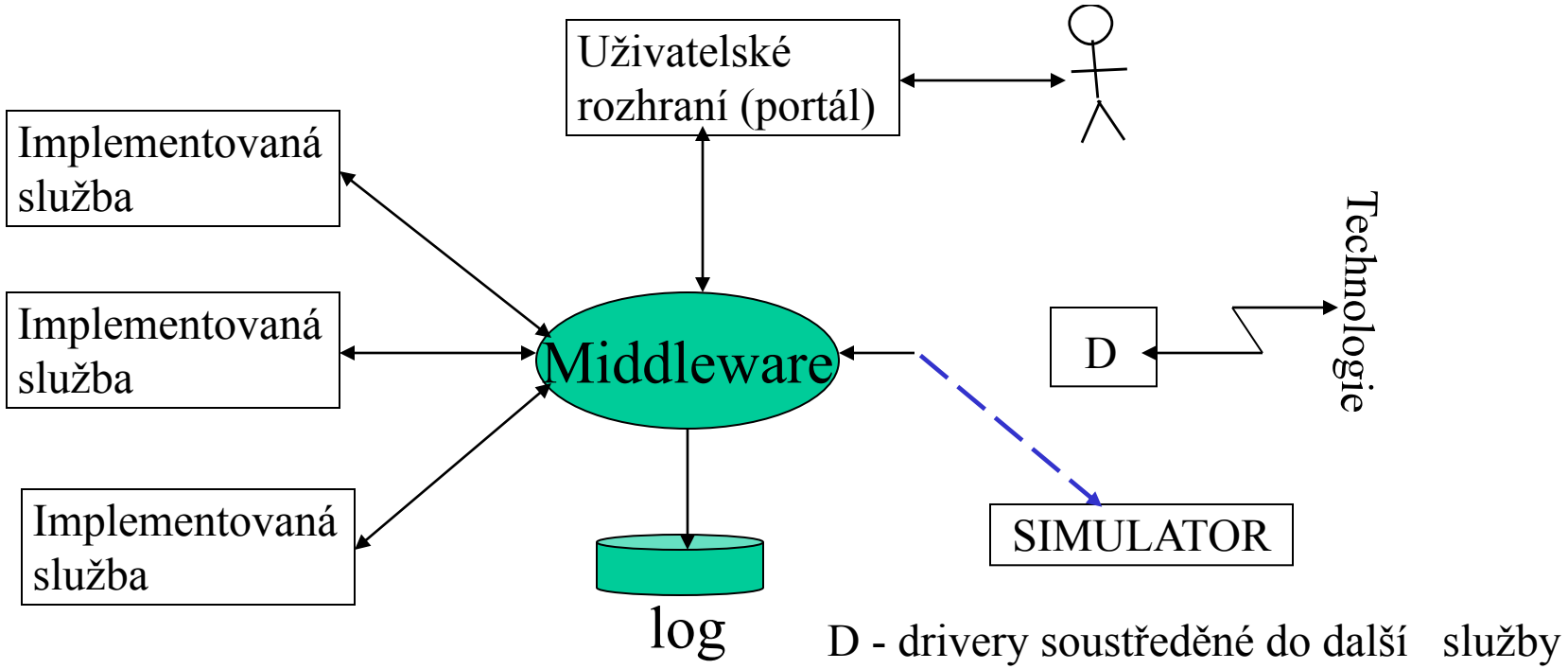
Techniky 2
Prototypování, ladění RT systémů



Zajímavý obrat – jak zjistit při dobu odezvy za nepřítomnosti řízené soustavy. Řešení

- Kalendář událostí v simulátoru,
- simulátor má nejvyšší prioritu

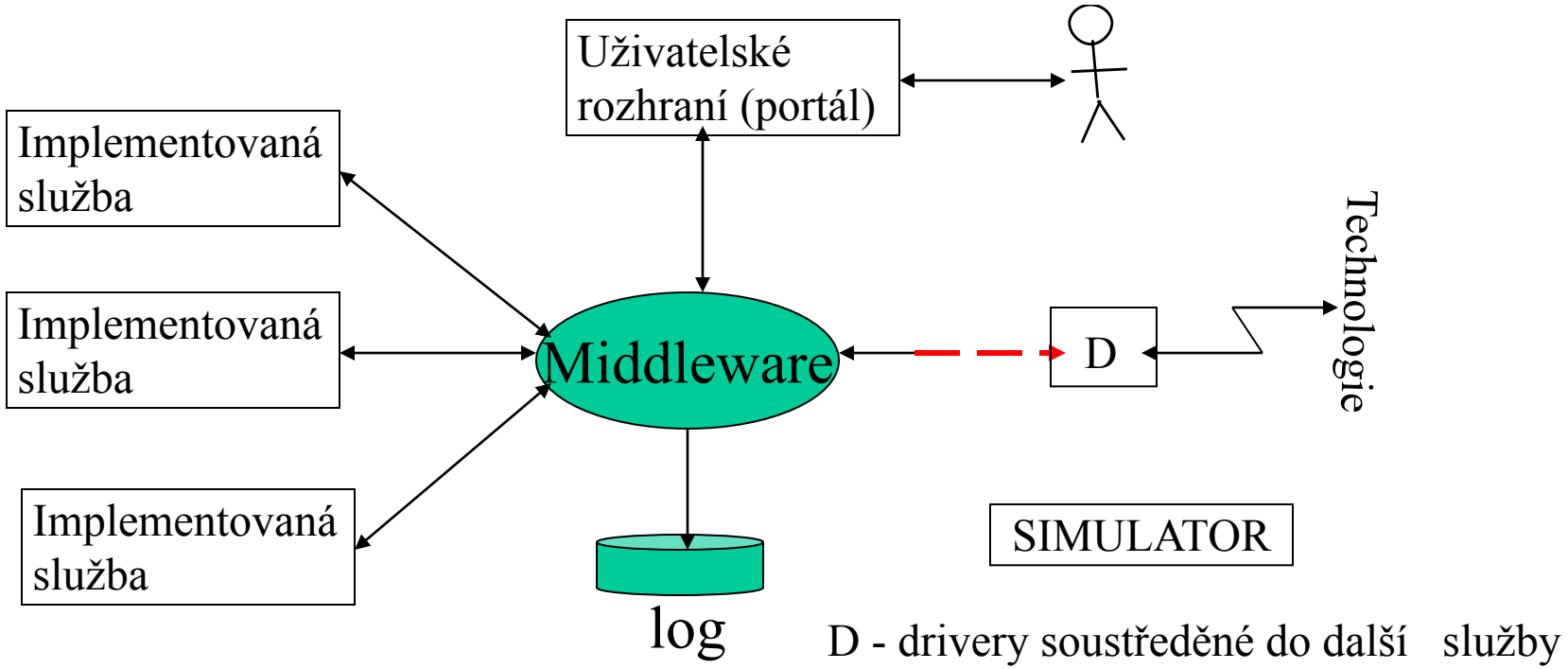
Techniky 2
Prototypování, ladění RT systémů



Zajímavý obrat – jak zjistit při dobu odezvy za nepřítomnosti řízené soustavy. Řešení

- Kalendář událostí v simulátoru,
- simulátor má nejvyšší prioritu

Techniky 2
Prototypování, ladění RT systémů



Zajímavý obrat – jak zjistit při dobu odezvy za nepřítomnosti řízené soustavy. Řešení

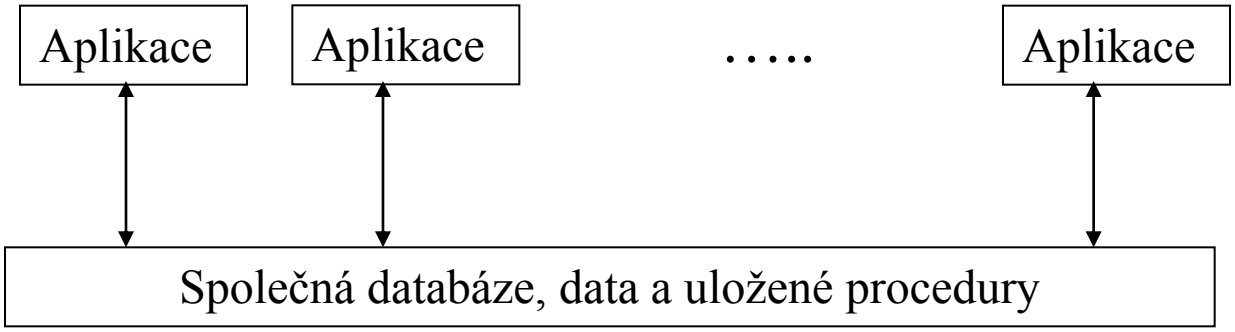
- Kalendář událostí v simulátoru,
- simulátor má nejvyšší prioritu

SOA podobné

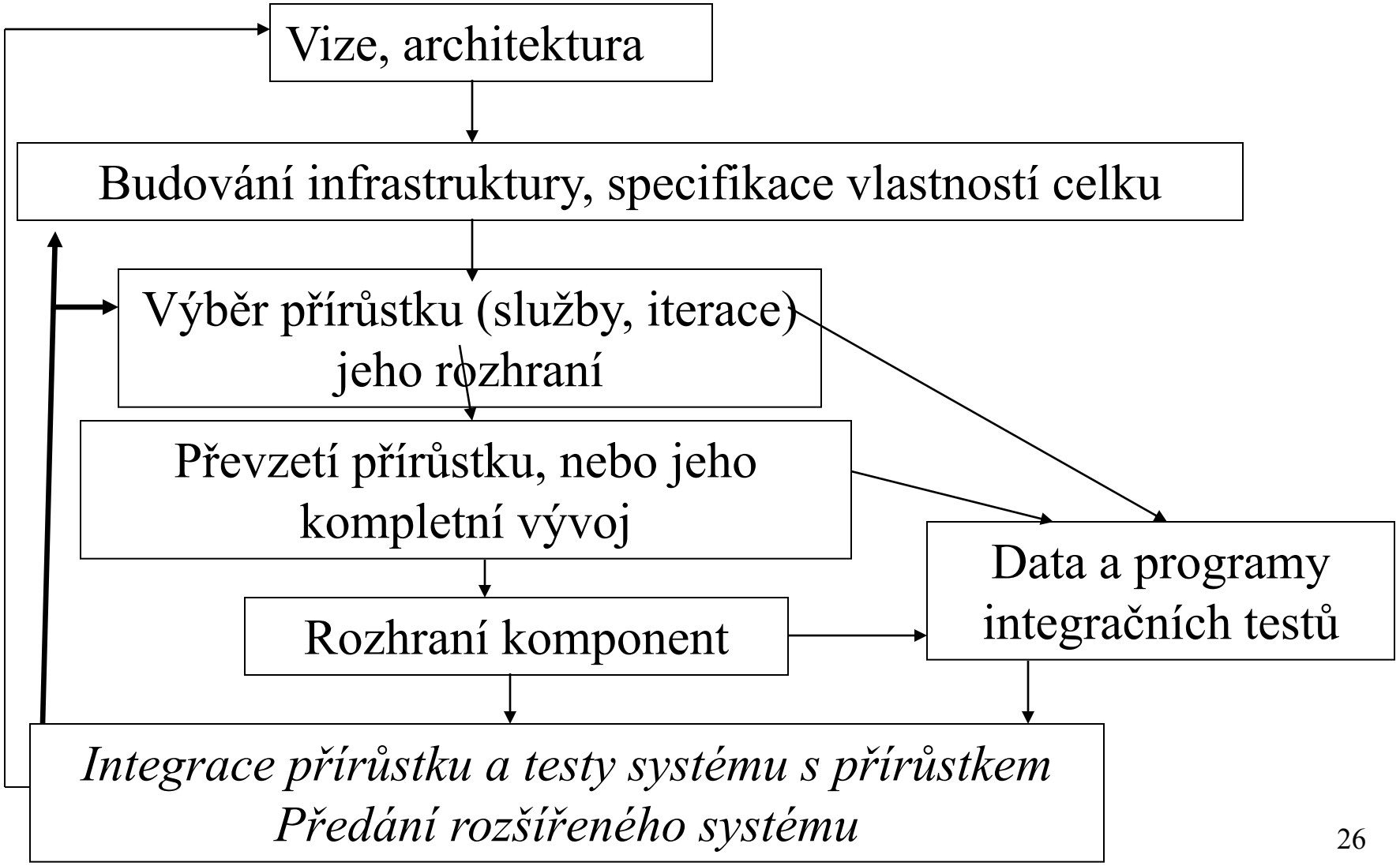
- Klasické SOA p2p síť pomocí meddlewaru přenášení zpráv
- Systém, kde funkce middlewaru nahrazuje centrální úložiště
  - Méně flexibilní a méně stabilní než p2p
  - Pro menší systémy s nutností společně DB dobře použitelné (banky)

SOA podobné

- Databáze nahrazuje middleware (poskytuje většinu jeho funkcí)
  - Výhodné pro implementaci systémů řízených událostmi, komunikace zápisem do DB
  - Lze kombinovat s výměnou zpráv, nedostatečně prověřeno



Inkrementální agilní vývoj



Co je rozhodující

- V SOA mohou být inkrementy jednotlivé služby
- Již prvé přiblížení nebo počáteční inkrement by měl poskytovat rozumnou funkčnost pro uživatele
  - Tedy nejen nějaké užitečné ale i pomocné funkce

Pojmy SWprocesů

- *SW proces* – síť kroků potřebných k vytvoření SW systému,
  - Varianta workflow pro vývoj SW
  - Obdoba byznys procesu
- *Krok procesu* – nedělitelná akce procesu
- *Prvek procesu* – logicky vázaná skupina kroků procesu
- *Agent* – aktivní entita (počítač, člověk)
- *Zdroje* - lidé, čas prostředky potřebné k provedení procesu nebo jeho prvku či kroku

Process engineering

Reprezentace procesního modelu. Použití částečně formalizovaných metod popisu a definice procesu, např. diagramů.

Analýza procesu. Prověrka zda model vyhovuje určitým kritériím (nadbytečnost kroků, úplnost, chybné řazení kroků atd.).

Instanciace procesu. Abstraktní model je parametrizován pro vývoj konkrétního produktu. Při tom se berou do úvahy technická organizační omezení a požadavky na vlastnosti produktu.(cosi jako customizace).

Process engineering

Provedení procesu (enactment). Provedení procesu může být úkolem lidí či softwarových nástrojů, nebo obou. Výsledkem provedení procesu je softwarový produkt.

Omezení provedení procesu. Proces a jeho kroky musí obvykle splňovat řadu podmínek, jako jsou povinnosti autorizace a dodržování standardů.

Podobné jako u jiných procesů

Životní cyklus procesu

Analýza požadavků na softwarový proces. Na základě vlastností cílového produktu a prostředí, vlastností týmu a prostředků, podmínek smlouvy se zákazníkem se zvolí základní vlastnosti procesu, např. inkrementální model.

Vývoj modelu. Cílem je vytvořit proveditelný softwarový proces. Vývoj modelu procesu obvykle zahrnuje plán tvorby modelu, volbu architektury procesu, návrh modelu, instanciaci a validaci modelu provedením inspekcí, případně ověřením na pilotním projektu. Často stačí instanciace existujícího modelu.

Přizpůsobení (tayloring). Adaptace modelu pro konkrétní projekt zpřesněním významu jednotlivých kroků a doplněním kroků.

Životní cyklus procesu

Plánování. Stanovení termínů provedení jednotlivých kroků procesu.

Instanciace. Doplnění údajů o agentech (kdo co kde provede) a zdrojích (co je k provedení třeba). U velkých projektů se připouští instanciace počátečních kroků procesu v situaci, kdy definice celého procesu ještě není dokončena.

Evoluce. V průběhu provádění procesu dochází obvykle ke změnám v důsledku nově vzniklých nebo nově zjištěných skutečností. To může ovlivnit definici softwarového procesu, případně i model procesu.

Provedení (enactment). Podle modelu, který nyní slouží jako plán činnosti, se uskuteční vývoj softwaru. Při tom se připouštějí úpravy modelu při výskytu neočekávaných skutečností.

Vlastnosti modelů – vize cílů

Věrnost (fidelity). Vlastnost vyjadřující, do jaké míry vystihuje potřeby a zavedené pojmy a procesy

Vhodnost (fitness). Vlastnost vyjadřující, do jaké míry je model vhodný pro daný účel. Důvody malé vhodnosti mohou být různé -- např. nepraktická doporučení nebo nedostatečná kvalifikace uživatelů, křivka učení.

Přesnost. Vyjadřuje správnost, úplnost a jednoznačnost modelu.

Vlastnosti modelů –technologické vlastnosti

Redundance. Vlastnost vyjadřující existenci takových kroků,které lze vynechat nebo nahradit jinými kroky. Redundantní kroky se používají pro definování alternativních postupů.

Škálovatelnost. Vlastnost vyjadřující rozsah velikosti projektů, pro něž je daný model procesu použitelný, resp jaké jsou možnosti budoucího rozšiřování.

Udržovatelnost. Vlastnost vyjadřující, jak snadno lze model a jeho instance modifikovat.

Vystopovatelnost. Lze najít důvody daného řešení

Vlastnosti modelů – kvalita řešení

Poněvadž je model softwarového procesu modelem paralelně pracujících aktivit, musí mít vlastnosti známé z teorie operačních systémů. Jsou to zejména:

Životnost. Tato vlastnost vyjadřuje, že při provádění nedojde k uváznutí (deadlock), tj. že nedojde k situaci, kdy proces nemůže pokračovat a přitom není řádně ukončen.

Robustnost. Vlastnost vyjadřující stupeň ochrany před neautorizovanými změnami a stability při vzniku neočekávaných situací.

Stabilita. Stupeň ochrany vůči nesprávným změnám a celkové spolehlivosti. Typickým příkladem je zvyšování stability izolováním změn instance od změn modelu.

Interaktivnost Stupeň autonomie a rozsah interakce s jinými procesy.

Podobnost s jinými typy procesů

- SW procesy jsou podtřídou business procesů – hlavní vlastnosti podobné
- Poznatky z SWP lze využít při specifikaci požadavků

Schéma vývoje procesu

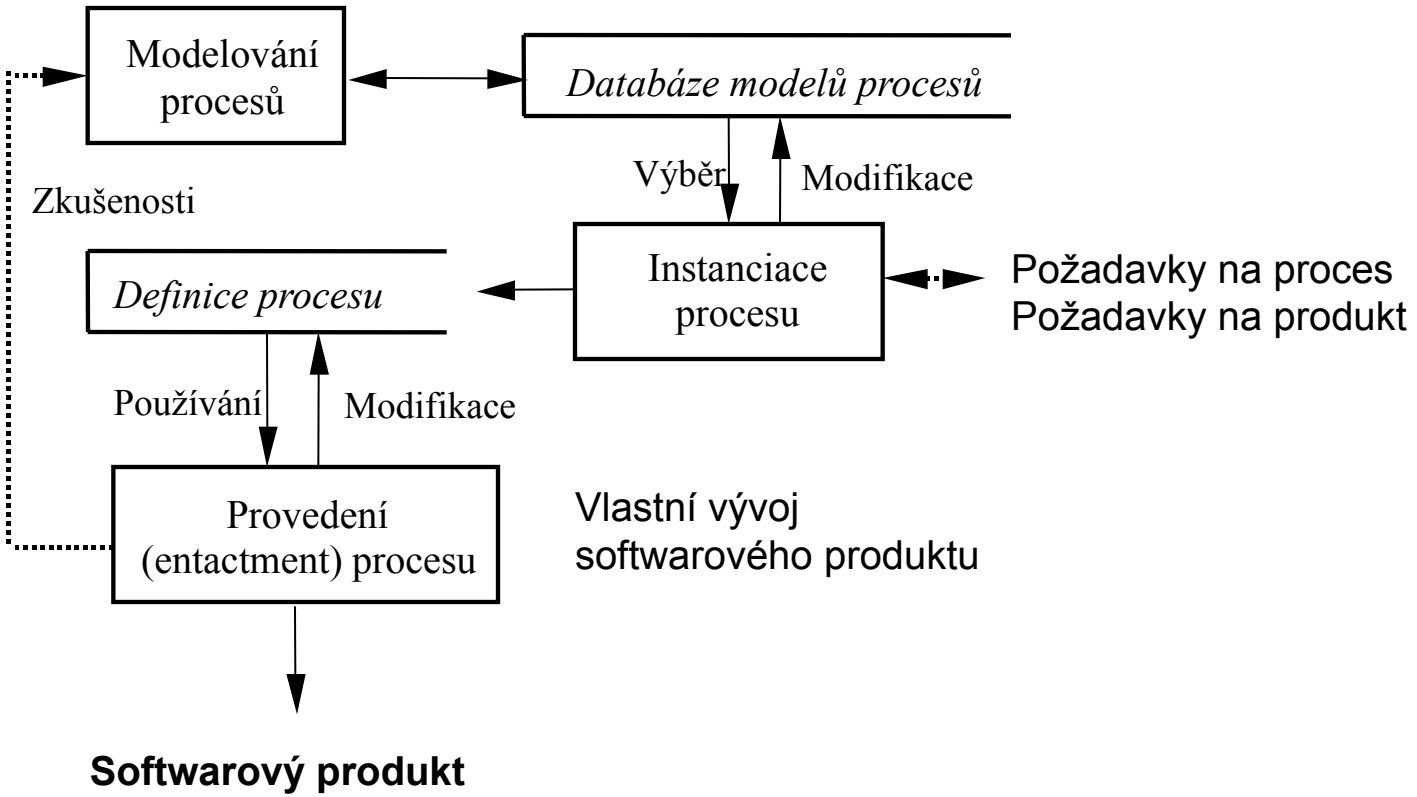
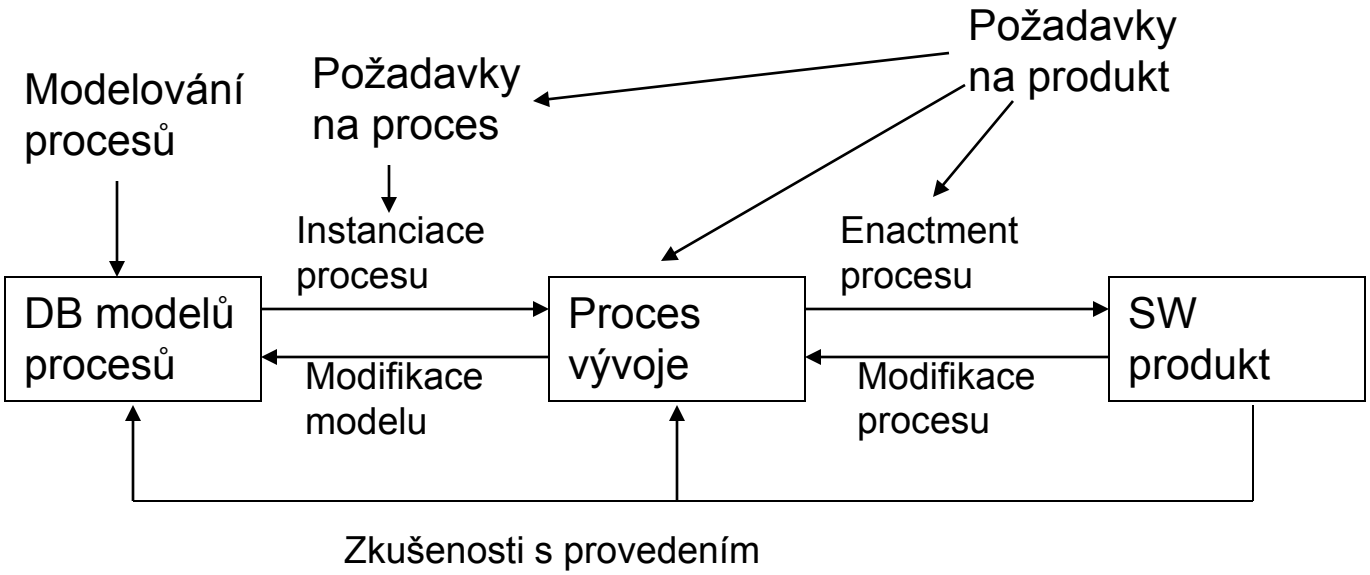
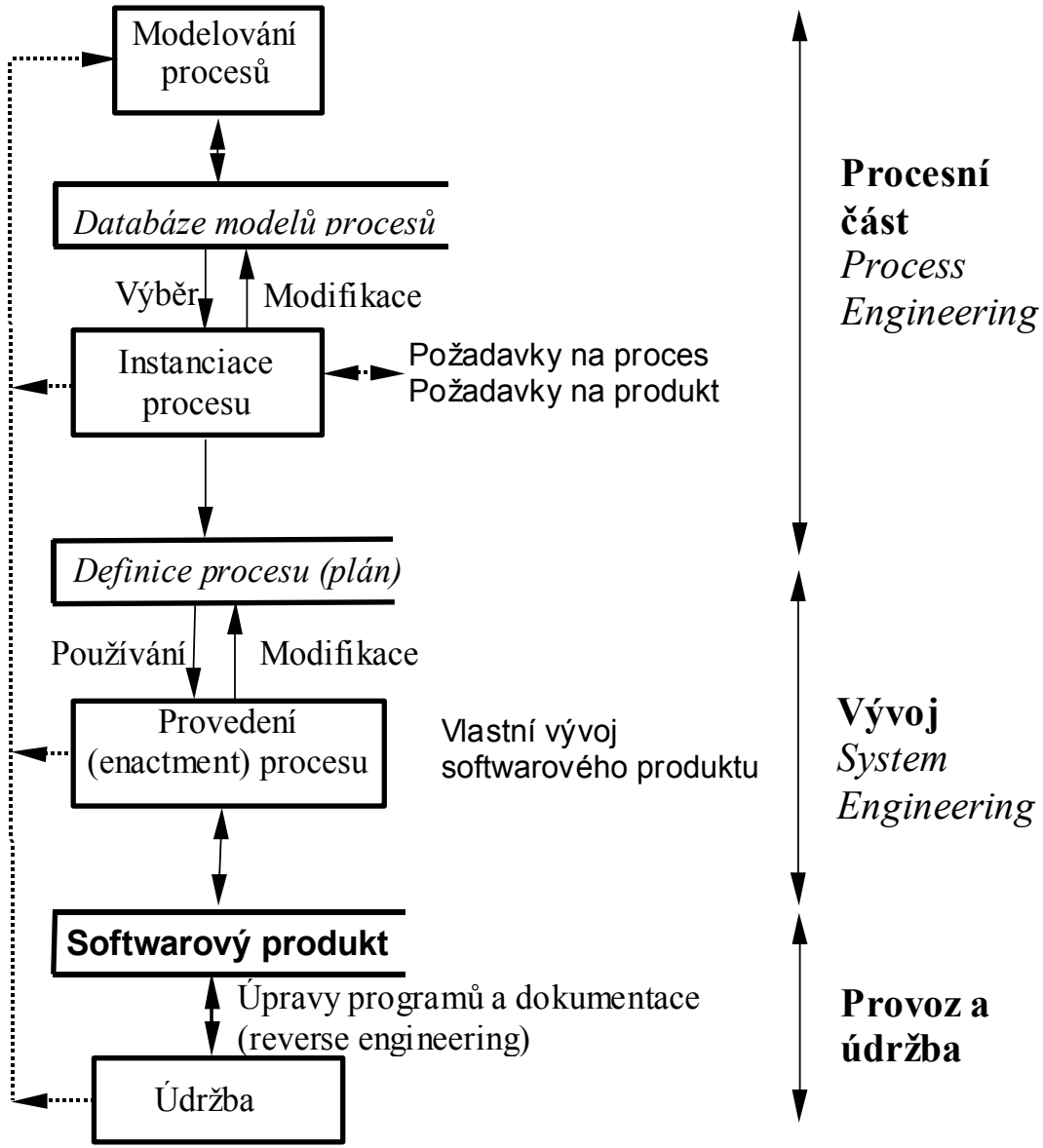
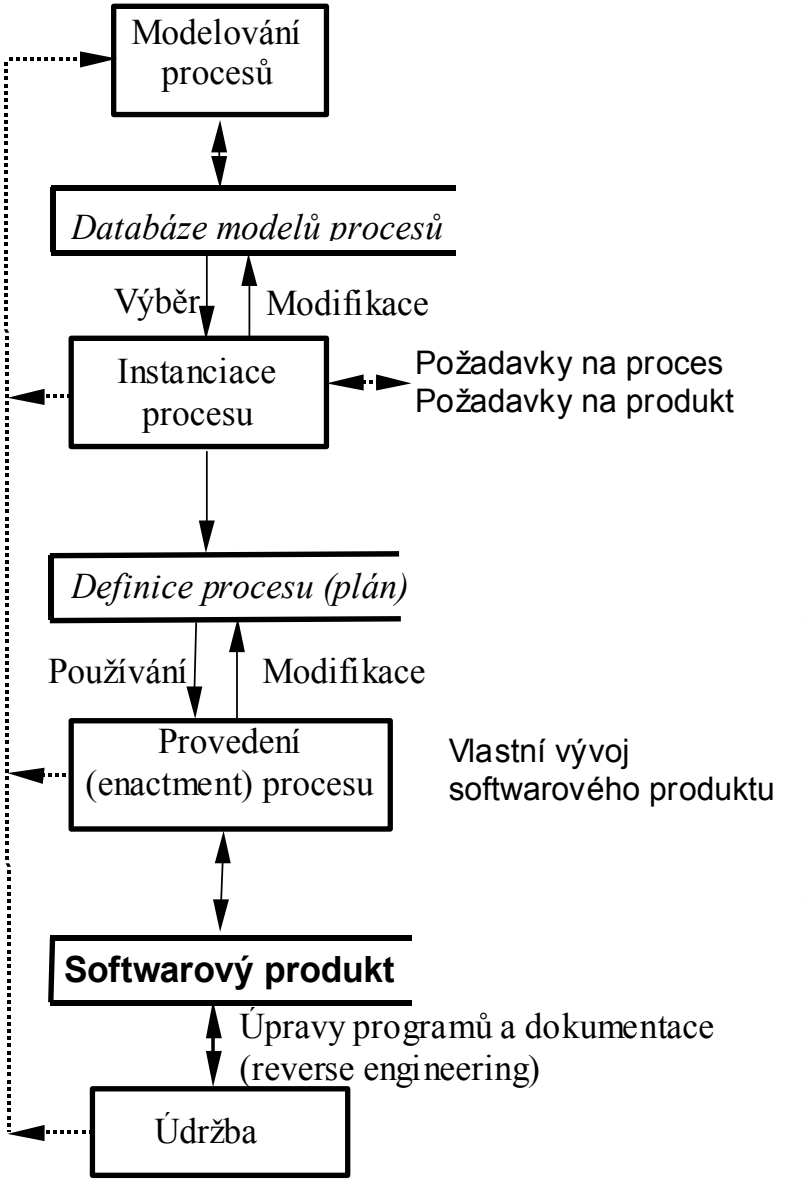


Schéma vývoje procesu



Použití procesů





Business architect

Systém architect

Procesní část
Process Engineering

Project manager
Process engineer

Použití procesů

Vývoj
System Engineering

System engineer

Provoz a údržba

Výše uvedené schéma lze použít i pro byznys procesy

- Obsahuje prvky agilního provádění a učení
- Pro malé podniky a také pro krátké série je to nutnost.

Prostředky popisu procesů

Kromě různých diagramů, mnoho z nich je převzato z UML se pro definici procesů používají dialekty vybudované nad UML

- Jazyky pro popis business procesů jako je BPDL, jazyk pro bussiness rules a jazyky pro definici pracovních toků. Jazyk pro provádění procesů BPEL
- Lze využít prostředky pro podporu řízení projektů jako je MS Project (ten ale používá CPMa s tím mohou být potíže), nebo některé funkce Lotus Notes případně i Excel

Překážky procesního přístupu

- Zavádění procesního přístupu při vývoji softwaru je spojeno s potřebou měnit u softwarové firmy zavedená pravidla hry. Procesní orientace nutí ke spolupráci jednotlivá dříve spíše samostatná oddělení. To může být spolu s tím, že softwarový proces je prostředek řízení, pociťováno jako přílišné omezování pravomocí a snižování prestiže vedoucích oddělení či týmů a vyvolat nepříznivé reakce. Je to obdobná situace jako v případě BPR (business process restructuring) při vývoji ERP.
- Procesní modely a jejich instance je třeba vytvářet ve spolupráci s vedoucími vývojových týmů a upravovat v průběhu řešení podle jejich připomínek (agilita).
- Modely procesů a jejich změny podléhají inspekcím a musí být schvalovány vývojáři, kteří jsou vlastně "uživateli" modelů.

Překážky procesního přístupu

- Nedostatečné znalosti lidí
- Problém specifikace procesů
- Přetěžování příliš rychlým zaváděním (příliš strmá křivka učení)
- Malá podpora vedení (srv. problémy s IS)
- Formální zavádění „na oko“
  - To je vůbec nejhorší

Zdokonalování procesů, Capability Maturity Model, CMM

1. Počáteční úroveň. Jak si kdo co pamatuje, když odejde, znalost se ztratí.
2. Opakovatelná úroveň. Standardy a DB v počátcích a na jednotlivosti. DB průběhu projektů. Není statistická analýza
3. Definované procesy. Standardy od specifikací požadavků až po managementu procesů a jejich provázanost.
4. Řízené procesy. Metriky – sběr a vyhodnocování, trendy, chybová místa, analýza s používáním matematické statistiky
5. Optimalizované procesy. Procesy neustálého vylepšování

Nezaručuje úspěch, lidé vždy klíčem, dobrým pomáhá, lemplové se mohou schovávat za papíry. Funguje spíše u větších podniků

Zdokonalování byznys procesů, Capability Maturity Model, CMM

Nezaručuje úspěch, lidé vždy klíčem, dobrým pomáhá, lemplové se mohou schovávat za papíry. Funguje spíše u větších podniků

Pravidlo: Vyšší úroveň zahrnuje vždy všechny nižší úrovně.

1. Počáteční úroveň

- SWP existují většinou jen v neformální formě a definují se případ od případu od počátku vždy znovu.
- Nejsou zavedena pevná pravidla plánování a řízení projektů.Výsledky vývoje závisí spíše na kvalitě jednotlivců než na kvalitě organizace práce.
- Zkušenosti se na celopodnikové úrovni se de facto nevyužívají, podnik jen v omezené míře zdokonaluje svou práci jako celek.
- Pokud nějaký pracovník z firmy odejde, jsou jeho zkušenosti pro firmu nenávratně ztraceny

2.Opakovatelnost

- Ve firmě jsou zavedena pravidla pro řízení projektů. Plánování a řízení projektů je založeno na zkušenostech s podobnými projekty, ale postupy se mohou případ od případu lišit. Řídí se však jednotnými zásadami platnými pro celou firmu.
- SWP nejsou plně standardizovány. Plány realizace jsou však realistické v důsledku využívaní předchozích zkušeností a je přísně sledováno, zda se dodržují. Při odchylkách od plánu jsou včas uskutečňována nápravná opatření.
- *Kritika: Ale to je možné jen při větších souborech dat, ty malé firmy nemají*

3. Definovanost

SWP jsou v rámci firmy standardizovány. Standardizace zahrnuje jak procesy softwarově inženýrské včetně specifikace a analýzy požadavků,tak manažerské.

Součástí norem jsou nástroje kontroly a zvyšování efektivnosti práce. Standardy jsou založeny na zkušenostech a na osvědčených softwarově inženýrských metodách a postupech včetně organizace a řízení prací,infrastruktury týmové spolupráce a školení pracovníků.

3. Definovanost

Součástí standardů jsou i procedury přizpůsobování SWP potřebám a zvláštnostem konkrétních projektů. Je zajištěna kontrola dodržování požadavků na funkce systému, nákladů a termínů.

Využívání SWP je založeno na odborném zázemí a na znalostech pracovníků firmy o aktivitách, rolích a odpovědnostech v SWP a podporováno skupinou vývoje a podpory SWP.

Znalosti pracovníků jsou rozvíjeny pravidelnými školením

4. Úroveň řízených procesů (controled level)

- Jsou definovány metriky kvality jak pro vyvíjený software, tak pro používané SWP.
- Je vyvinut systém sběru, sledování a vyhodnocování metrik jednotným způsobem v rámci celé organizace využívající celopodnikový IS metrik.

4. Úroveň řízených procesů (controled level)

- Vyhodnocování dat je schopno odlišit náhodné fluktuace od statisticky významných změn.
- Firma je schopna vyhodnocovat trendy a odhadnout hodnoty důležitých metrik, jako jsou termíny a náklady. Je schopna odhadnout i přesnost odhadů stanovením konfidenčních intervalů, tj. mezí, do nichž s velkou pravděpodobností padne odhadovaná hodnota.

Úroveň optimalizovaných procesů (optimized level).

Jsou zavedeny procedury neustálého vylepšování SWP.

Je vytvořen tým hodnotící kvalitu procesů a navrhující jejich vylepšování včetně zavádění nejnovějších metod, postupů a nástrojů. Tým analyzuje příčiny úspěchů i neúspěchů a zobecňuje získané poznatky. Odlišuje při tom náhodné od zákonitého.

Metodika málo analyzuje meze statistiky, a problém malých souborů dat (to ale platí i pro COCOMO)

Pro malé podniky je používání úrovní 4 a 5 omezeno malým rozsahem dat, omezení jsou i pro úroveň 3

Úroveň optimalizovaných procesů (optimized level).

Na základě analýz modifikuje používané SWP. Zlepšení se realizují formou dílčích změn SWP i jako zásadní inovace využívající nové metodologie a technologie.

2. Opakovatelnost

- Řízení a kontrola specifikace požadavků
- Plánování na základě SWP
- Dohled na SWP a jejich archivace
- Řízení subkontraktů.
- Zajišťování kvality.
- Řízení konfigurace.

2.Definovanost

- Koordinace a standardizace SWP v rámci organizace.
- Standardizace prací všech etap vývoje SW.
- Programy školení.
- Integrované řízení vývoje SW a SWP
- Podpora týmové práce a spolupráce mezi týmy.
- Audity.
- Práce týmu podpory SWP.

4. Úroveň řízených procesů (controled level)

- Definice metrik SWP a systém jejich využívání.
- Kvantitativní metody řízení a plánování využívající metody
  statistické analýzy dat.
- Komplexní metody řízení kvality
-

Úroveň optimalizovaných procesů (optimized level).

- Prevence závad.
- Řízení postupných změn metod a praktik.
- Optimalizace softwarových procesů porovnáním variant procesů s využitím protokolů provedení procesů v minulosti
- Stálý tým analýzy softwarových procesů a jejich rozvoje

Úrovně 4. a 5. a do značné míry i 3. jsou dosažitelné jen u velkých organizací.

Problém dostatečně velkých souborů dat.

Nebezpečí byrokratizace vývoje softwaru.

CMMI, COBIT,ITIL

- CMMI integrated – integrace CMMI do celkového řízení projektů – tlustá kniha doporučení
- COBIT – zapojení principů CMMI do budování souborů služeb
- Opět omezené možnosti uplatnění u malých firem
- ITIL IT infrastructure library, soubor služeb, přednostně v e-gov

Agilní formy vývoje

Existuje řada aplikací, které jsou malé, nekritické a mohou být použity i jako doplněk kritického jádra (př. analýza finančních dat)

Na ty je vhodné použít agilní formy vývoje.

Hlavní zásady, agile manifesto

- **Individuals and interactions** over processes and tools
  Working software over comprehensive documentation
  Customer collaboration over contract negotiation
  Responding to change over following a plan

Zásady práce

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.

Zásady práce

- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Zásady práce

- Continuous attention to technical excellence and good design enhances agility.
- Simplicity--the art of maximizing the amount of work not done--is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams (my to nazýváme demokratický tým).
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Hlavní principy agilního vývoje

- Systém je realizován po malých kouscích (většinou aplikacích) - iteracích
- Jedinou finální dokumentací je program sám, měl by být samodokumentující
- Vývojový cyklus je tak krátký, že nejsou potřeba prototypy a otestují se i specifikace
- Rotace rolí

Agilní vývoj omezuje rizika

- rizika spojená s nepřesným zadáním, resp se složitostí bydovaného systému
- rizika spojená s fluktuací členů týmu,
- rizika spojená s tím, že neexistuje dokumentace v obvyklém rozsahu,
- rizika spojená s nedodržováním termínů a překračováním rozpočtů.

Tým při agilním vývoji

- Do 10 členů.
  - Kouč
  - Programátoři
  - Časoměřič
  - Stále přítomný pracovník uživatele (asi nejde vždy dodržet, mohou být třeba různí pracovníci, daný pracovník je nepostradatelný)
  - Programátoři pracují ve dvojicích, které se mění
    - Prvý programátor – vymýšlí a píše
    - Druhý programátor – oponuje, kontroluje, spoluvymýšlí
- Místnost pro odpočinek a jednání

Průzkum

- Celý systém je vyvíjen v řadě malých kroků zvaných iterace. Každá iterace začíná specifikací požadavků za účasti zákazníka (tzv.user story) a má být implementovatelná během několika málo dní.
- Tím se dostane seznam úkolů – kandidátů na zařazení do dané iterace. Následuje „plánovací hra“, při které programátoři nejprve pro jednotlivé funkce a modifikace funkcí, jejichž implementace se v dané iteraci zvažuje, odhadují, kolik času bude na implementaci potřeba. Cílem je odhad, kolik času si vyžádá daná iterace. Nemělo by to být více než tři týdny, ideální je jeden týden. To je obvykle možné jen tehdy, zredukuje-li se počet úkolů
- Z dané iterace se vyřazují úkoly s nejmenší prioritou, tj. s nejmenším přínosem.

Průzkum

- Priority stanovuje zákazník podle schématu:
  - daná funkce je velmi potřebná/nutná,
  - tuto funkci by bylo velmi žádoucí mít, ale zatím se lze bez ní obejít,
  - funkce by se hodila, její přínos a potřeba nejsou v daném okamžiku jednoznačné.
- Postupuje se při tom tak, že zákazník jakoby přesvědčoval programátory, proč je má daná funkce určitou prioritu (jedná se o jistý druh oponentury, vyžaduje to však dobré vztahy mezi zákazníky a programátory). Výsledkem je odsouhlasený seznam úkolů dané iterace spolu s odhady jejich časové náročnosti a určením dvojic programátorů, které budou jednotlivé úkoly řešit.
- Vypracuje se a oponuje plán realizace iterace

Implementace

- Programování ve dvojici. Každý programátor může upravovat libovolnou část systému (pokud možno)
- Začíná se návrhem testů a pak se teprve programuje
  - A co když nevím co nevím, že je správně
- Testy se integrují s testy ostatních částí a použijí se při testování nových částí i při jejich integraci
- Neustále se kontroluje dodržování plánu
- Do dvou týdnů dokončit a předvést, lepší inkrementální postup

Přehled pravidel

- *Práce v týmu.* Vývoje se realizuje v týmu, jehož členy jsou programátoři, kouč, časoměřič a uživatel.
- *Malé kroky.* Vývoj je dekomponován do malých kroků – iterací. Každá iterace začíná výběrem úkolů tak., by mohla být dokončena nejdéle do třech týdnů. Iterace končí akceptací nových funkcí.
- *Jednoduchost.* Systém navrhujeme co nejjednodušší, jak pro splnění plánovaných úkolů možné. Nepotřebné komplikovanost se odstraňuje ihned, jak se zjistí.

Přehled pravidel

- *Nepřetržité plánování a kontrola plnění plánu.* Základem plánu je výběr úkolů a odhad jejich pracnosti těmi, kteří úkoly řeší. Plán se neustále porovnává se skutečností a navrhuji se nápravné akce, případně se plán aktualizuje.
- *Programování.* Programování se provádí ve dvojicích. Dvojice se formují pro každou iteraci znovu a úkoly dvojice se mohou týkat libovolné části systému. Obvyklý nástroj programování jsou CRC karty.
- *Testování.* Testy částí se píší programátoři před tím, než se začne příslušná část programovat. Současně uživatelé připravují testy funkcí. Testy se integrují do systému automatického provádění testů. Testy se spouštějí co nejčastěji. Úkoly související s provozem subsystému testů mohou být formulovány jako dvojic.

Práce ve dvojici je často výhodná

- Čtení kódu
- Vedoucí dvojice týmu
  - Vedoucí dvojice vše dělá
  - Druhý mu oponuje, hledí pod prsty
  - Učí se od něj
  - Může ho vždy zastoupit

Přehled pravidel

- *Refaktorizace.* Programy se modifikují z důvodů zjednodušování programů, odstranění opakování kódu nebo s cílem zlepši SW inženýrské vlastnosti kódu. Refaktorizace se považuje za běžnou praxi a provádí se často.
- *Společné vlastnictví.* Každý člen týmu je schopen měnit libovolnou část programů. Každý cítí odpovědnost za celek.
- *Častá integrace.* Sytém je integrován i několikrát denně po dokončení libovolného úkolu, integrace je spojena s provedením testů.

Přehled pravidel

- *Málo přesčasů.* Zpravidla se nepracuje více než 40 hodin týdně, Pokud jsou v některém týdnu přesčasy, nesmí být žádné přesčasy v týdnu následujícím.
- *Intenzivní společenský život* (čas na oslavy úspěchů, je dobré si občas společně zasportovat)
- *Zákazník.* K dispozici týmu má být k dispozici koncový uživatel, který formuluje úkoly a odpovídá na otázky.
- *Standardy pro psaní program.* Dodržují se dohodnutá pravidla pro psaní programů. Pravidla jsou zaměřena na to, aby programy mohly sloužit jako prostředek komunikace mezi členy i nečleny týmu.
- *Modernizace* znalostí, technik, metod, odborný růst členů týmu

Výhrady k agilnímu vývoji

- Zaměřeno příliš na vývoj od počátku s výsledkem spíše jedna aplikace/program
  - A tedy ne SOA, to je věcí dalšího výzkumu
- Málo řešena integrace hotového SW
- Nevhodné pro
  - Kritický SW
  - Velké systémy
  - SW typu COTS (?)

Kdy to nejde

- Kritické systémy, kde je nutné přesně dodržovat dohodnuté (technologie)
- Rozsáhlé systémy, které se nedají dobře dekomponovat
- Nejsou k dispozici kvalitní řešitelé
- Není ochota se domlouvat o cíli za pochodu (jak uzavřít smlouvu, jak sankce za neplnění)
- Systémy kustomizované ?

ISO normy pro SW procesy

- **ISO/IEC 12207:1995** Information technology -- Software life cycle processes, modernizuje se v ISO 20000
- **ISO/IEC 12207:1995/Amd 1:2002**
- **ISO/IEC 12207:1995/Amd 2:2004**
- ISO/IEC TR 15271:1998 Information technology -- Guide for ISO/IEC 12207 (Software Life Cycle Processes)
- ISO/IEC TR 14759:1999 Software engineering -- Mock up and prototype -- A categorization of software mock up and prototype models and their use
- **ISO/IEC 15504:2004** also known as **SPICE** (Software Process Improvement and Capability dEtermination) is a "framework for the assessment of processes"
- ISO/IEC 90003:2004Software engineering -- Guidelines for the application of ISO 9001:2000 to computer software

ISO 20000 2005

- Řízení procesů vývoje softwaru
  - Management procesů
- Hodnocení kvality SW procesů
- Vyšel český překlad (Marie Šebestová a ostatní)

ISO 20000, IT service management

- Home ISO 20000 The Contents The Benefits ISO 20000 Download ISO 20000 & ITIL Contact Page
- **What Is ISO 20000?**
- ISO 20000 is the international standard for IT Service management.
- The standard actually comprises two parts: ISO/IEC 20000-1 and ISO/IEC 20000-2. ISO 20000-1 is the 'Specification for Service Management, and it is this which is certifiable against. ISO 20000-2 is the ' Code of practice for Service Management', and descibes best practices, and the requirements of Part 1.
- **What Was BS15000?**
- ISO 20000 is in fact based upon an original pair of documents, BS15000-1/2, which were published by BSI in 2002 and 2003 respectively. An earlier version of BS15000-1 was first published in 2000. Even this, however, was not the earliest iteration. As far back as the 1980's a BSI group called the 'Service Management Group' was at work defining ITSM processes.
- An example fo this work is provided by the following diagram, which illustrates the state of play in 1998:
- **The Standard Evolves**
- By the time the new release of BS15000 was published in 2002, the framework had been harmonized with other international standards, to embrace the familiar PDCA (Plan-Do-Check-Act). This approach is illustrated below:
- The scene was thus set for ISO 20000, which was published at the end of 2005.
- **ISO 20000 Resources**ISO 20000 Central is designed to provide a range of information to support the standard. In addition, a number of support resources have been identified. These, as well as several sources of the standards themselves, can be viewed via the selections on the right hand side.
- (c) ISO 20000 Central 2005

ISO 20000, IT service management

- Home ISO 20000 The Contents The Benefits ISO 20000 Download ISO 20000 & ITIL Contact Page
- **What Is ISO 20000?**
- ISO 20000 is the international standard for IT Service management.
- The standard actually comprises two parts: ISO/IEC 20000-1 and ISO/IEC 20000-2. ISO 20000-1 is the 'Specification for Service Management, and it is this which is certifiable against. ISO 20000-2 is the ' Code of practice for Service Management', and descibes best practices, and the requirements of Part 1.
  What Was BS15000?

ISO 20000, IT service management

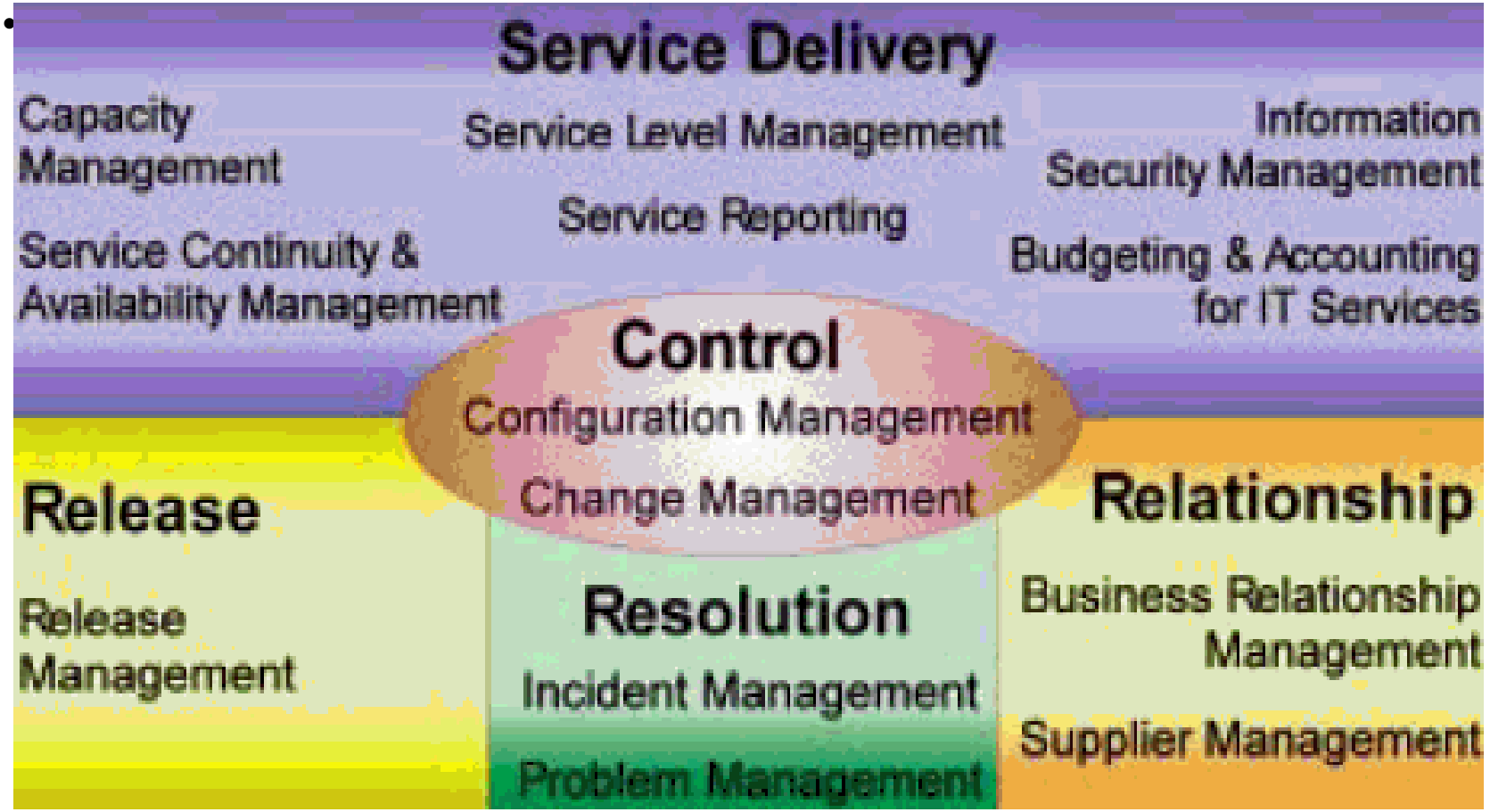
- ISO 20000 is in fact based upon an original pair of documents, BS15000-1/2, which were published by BSI in 2002 and 2003 respectively. An earlier version of BS15000-1 was first published in 2000. Even this, however, was not the earliest iteration. As far back as the 1980's a BSI group called the 'Service Management Group' was at work defining ITSM processes.

ISO 20000, IT service management

- **The Standard Evolves**
- By the time the new release of BS15000 was published in 2002, the framework had been harmonized with other international standards, to embrace the familiar PDCA (Plan-Do-Check-Act). This approach is illustrated below:
- The scene was thus set for ISO 20000, which was published at the end of 2005.

ISO 20000 ResourcesISO 20000 Central is designed to provide a range of information to support the standard. In addition, a number of support resources have been identified. These, as well as several sources of the standards themselves, can be viewed via the selections on the right hand side.

ISO 20000, IT service management



ISO 20000, IT service management