Kódování a testování

Před předáním

Kódování

Psaní programu podle přesných specifikací

Člověk jako kompilátor

Kódování

- Kódování není dnes hlavní problém
- Jedna sedmina pracnosti vývoje a pracnost kódování se dále zmenšuje
  - Ví se, jak na věc a jak to učit a standardizoat
  - Systémy podpory programování (vizuálnost, CASE, Delphi, Power Builder, …), asi nevhodné pro SOA
  - Servisní orientace, objektová orientace, komponenty
  - Problém je ve specifikacích, to je úzké místo, kódování nikoliv
- Při kódování je výhoda mládí. Nebezpečí, že se mladí omezí na kódování a nic dlouhodobějšího se nenaučí

Programovací jazyky

- Jsou méně důležité, než se má za to, úzké hrdlo je jinde (Prof.Malík simuloval lidské srdce ve Fortranu, ač se tento jazyk pro to nehodí, stálo ho to dva měsíce práce, koncepce byla dobrá díky jazyce Simula, jazyka pro programování simulací. Simulátor byl totiž nejprve napsán v jazyce Simula .Simulátor se používal při testech správnosti aplikace kardiostimulátorů a jejich vylepšování)
- Nejen vlastnosti jazyka, ale také vývojového prostředí, včetně knihoven a hotových částí, zásuvné moduly
- Důležité, jak se jazyk uplatní v podpoře SW architektur (COBOL a dataflow diagramy pro dávkové výpočty)
- Nový jazyk se zpravidla uplatní jen, je-li určen pro volnou niku na trhu (Java pro webové stránky, srv. FORTRAN kontra Algol či PL/1)
- Znalosti vývojářů závisí na jazyce a s ním spojenými nástroji a metodikami

Testování

- Součást evaluace (ověřování), zda produkt odpovídá potřebám uživatelů
- Nejpracnější etapa vývoje
- Jde automatizovat jen zčásti. Důvody:
  - Testuje se i správnost specifikace a ta je fuzzy a mění se v čase, blokované znalosti
  - Nutné testovat předpoklady o schopnostech a potřebách uživatelů
- To vyžaduje spoluúčast uživatelů

Testování

Mnohé problémy lze automatizovat jen zčásti i v případě dokonalých specifikací, jaké bývají u kritických aplikaci (selhání je podtatný problé).

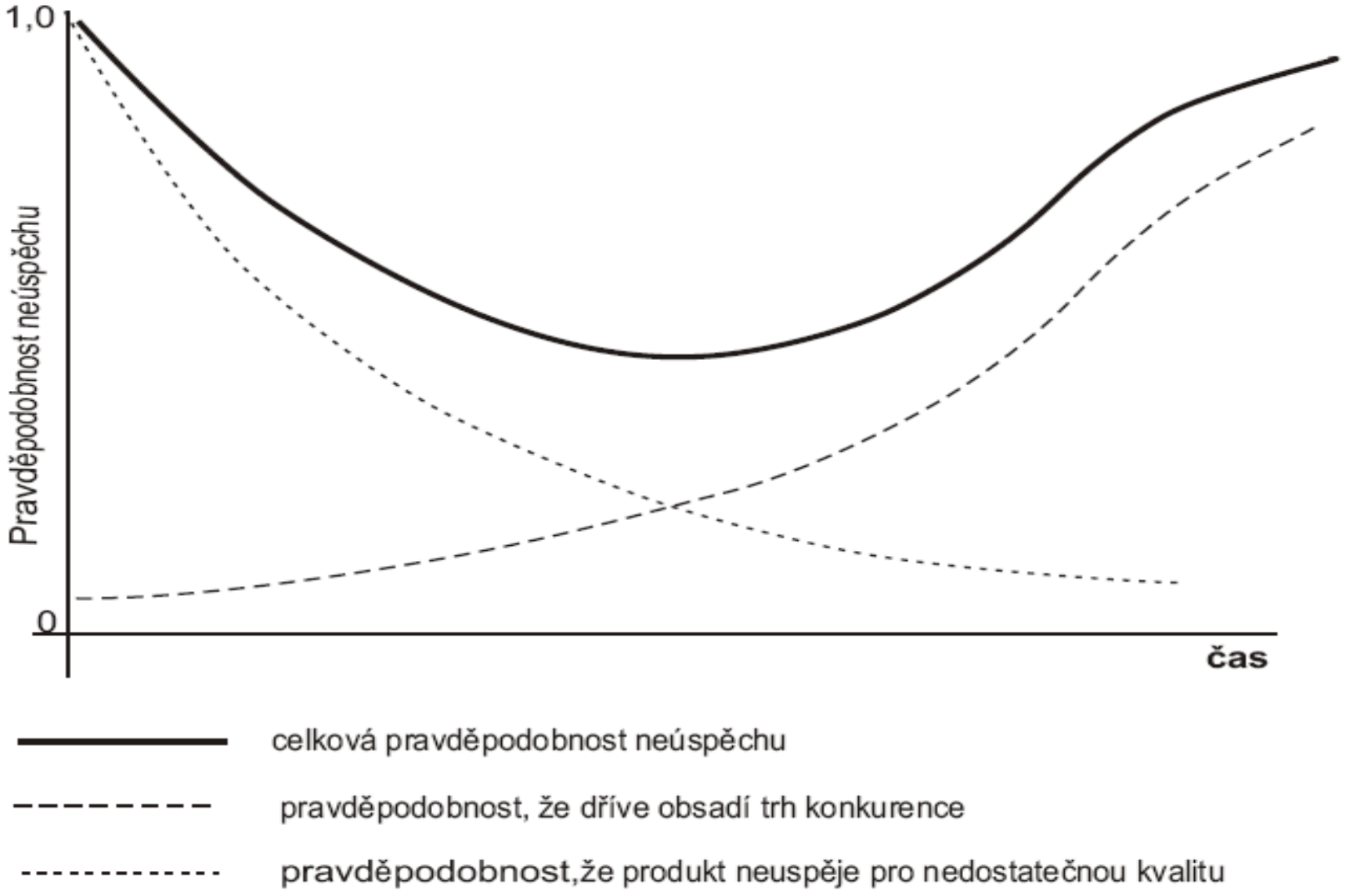
Důvody: Churchova téze, co nejde algoritmizovat na Turingově stroji, nelze vůbec, složitost reálného světa – ten není počítačem, je náhodný v principu (kvantová mechanika) i díky své složitosti

- Algoritmicky nerozhodnutelné problémy při testování, na příklad
  - Detekce mrtvého kódu
  - Otestování všech kombinací návazností větví programu
  - Detekce nekonečného cyklu
- Důsledek: Nelze plně automatizovat, tvůrčí problém,
  - Řeší se heuristicky, vždy ale nějaký problém zůstane
  - Často lze řešit přibližně (lineární programování, rozvrhovací algoritmy)
  - Zakopaný pes je často v nesprávném odhadu potřeb a přínosů při specifikaci potřeb

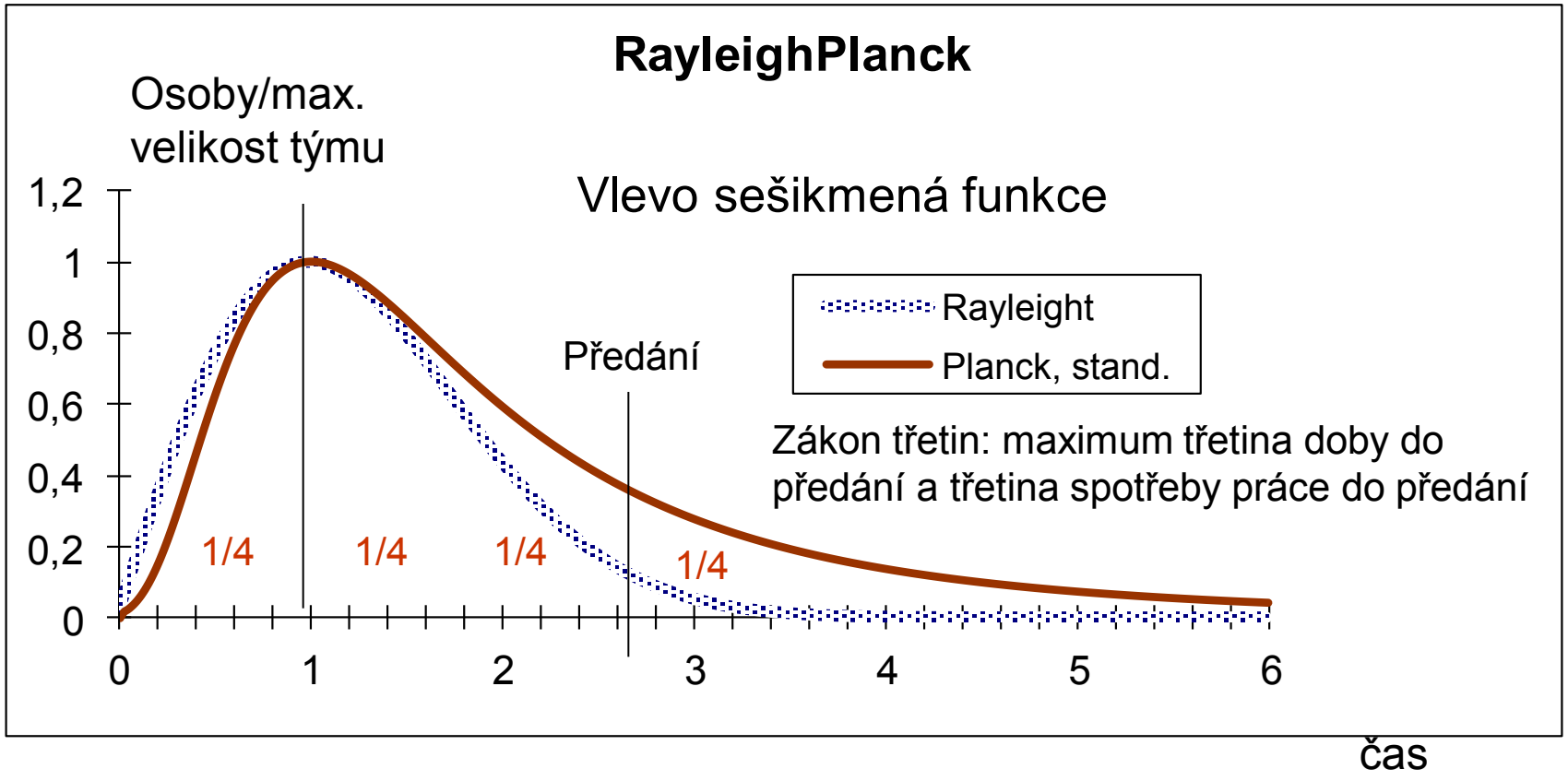
Testování

- Součást evaluace (ověřování), zda produkt odpovídá potřebám uživatelů
- Většinou zahrnuje i testování s uživatelem
  - To je největší problém
- Výše uvedené problémy indikují, že nelze prakticky nikdy odstranit všechny problémy, nelze zero defect software

Pravděpodobnost neúspěchu v závislosti na době testování



Najímaný tým, vrchol a odhad doby řešení



Transformace proměnných tak, aby max bylo v bodě 1 a mělo hodnotu 1 a v nule byla hodnota funkce prakticky nula. U pevného týmu odpovídá intensitě práce

Nelze zero defect software

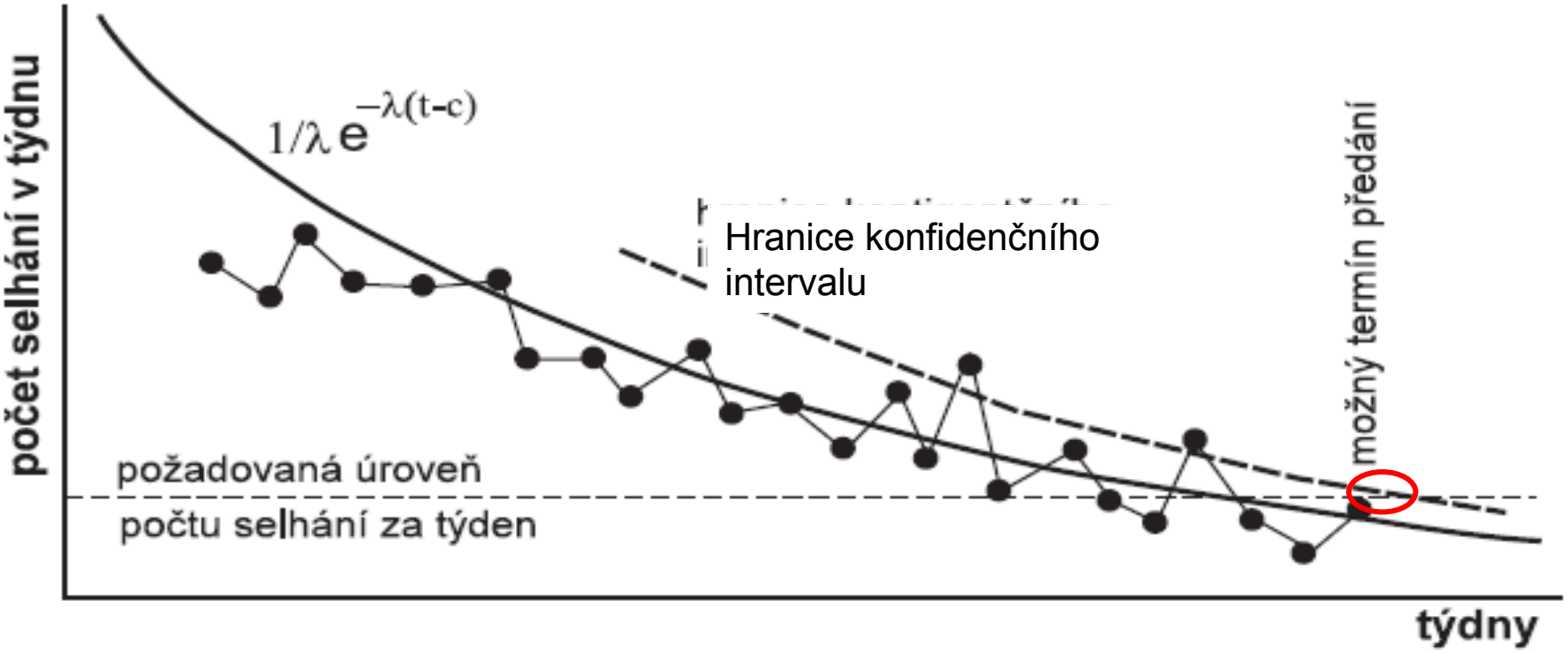
Ale jak řídit, když průběh funkcí neznám?

- Intuice manažera - odhadne nejvhodnější dobu
- Obecně je pocit, že lépe předat před minimem než po minimu
- Proč?

Obecně je pocit, že lépe před minimem než po minimu

- Po minimu z dlouhodobého hlediska zvětším možnost, že vyroste konkurence
- Také ne zcela správný pocit, že už je to dobré

Jak poznat, že je produkt dostatečně spolehlivý, použitelné jako test ukončení testování u kritických aplikací



Existují účinnější postupy z teorie spolehlivosti.

Exponenciální model poklesu frekvence selhání je zpravidla správný (podmínkou je nezávislost detekce jednotlivých selhání)

Druhy testů

- **Částí, (unit tests)**
  - samostatných kusů programů
  - dost často testují programátoři sami
- **Integrační**
  - Shora
  - Zdola
  - Selektivní, sendvič, jádro a programy
- **Regresní (zopakování většiny testů)**
  - Může být příliš náročné na uživatele a na provoz, v agilním vývoji spíše pravidlo

Druhy testů

- **Funkcí**
  - Ucelených akcí systému
- **Systému**
  - V simulovaném provozu jako celek
- **Předávací**
  - Podle smlouvy
- **Test užíváním**
  - Zkušební provoz
- **Test simulací nebo prototypem**

Integrace zdola

- Je třeba mnoho pomocných dat a programů
- Funkce systému se testují a mohou předvádět poměrně pozdě

+ Moduly jsou obecněji použitelné (méně závisí na změnách funkcí systému)
+ Ověřují se možnosti implementace

Integrace shora

+ Je třeba méně pomocných dat a programů
+ Funkce systému a rozhraní se testují a mohou předvádět poměrně brzy
- Moduly jsou použitelné jen v daném prostředí (někdy je to výhoda)
- Chyby na vyšších úrovních mohou být fatální (až příliš pozdě se zjistí problémy s implementací)
- Problém s centralizací požadavků. Centrální prvek je hrozba i zdroj potíží

Podpora testů

- CASE systémy mají testovací roboty
  - IBM Rose, RRrobot
  - Agilní přístup, buduje se testovací podsystém
  - Generátory (power builder)
  - Systémy podpory programování usnadňují testování, spíše detailů
- Pracnost testování závisí na architektuře systému, v SOA a při agilním vývoji je menší
  - Znovupoužitelnost a produkty třetích stran
  - Možnost testování služeb přesměrováním zpráv
  - Využívání prototypů a žurnálů
  - Mentálně zvládnutelné
  - Použití ISO norem u kritických aplikací výhodiu i nutnosti
  - Pozor: Integrace existuuících komponent velkých komponent (sdlužeb) je spíše vývoj zdola.

Programátoři a testéři

- Tři varianty „spolupráce“
  1. Programátor je současně testér
     - Populární, rychlé, málo účinné (vadí u kritických aplikací)

       Kompromis: Unit tests. Testy částí.
  2. Testér je specifická role, bílé skříňky
     - Testér spolupracuje s programátorem při nápravě selhání
  3. Testér testuje černé skříňky, testéři nemají zdrojové kódy ani kontakt s programátory
- Nejúčinnější je 3, ale je to velmi drahé a vyžaduje to kvalitní profesionály

Terminologie testování

- **Selhání** – jiný než očekávaný výsledek
- **Neúspěch testu** – nedetekuje selhání
- **Testový případ**: Data, scénář, výsledky
- **Testová procedura**: Síť testových případů
- **Test**: Síť testových procedur
- **Položka k testování**: Vše, co potřebuje testový případ (prostředí, SW, data, scénáře, očekávané chování systému, výstupy)

Terminologie testování

- U agilního programování se nejprve definují testové případy pro nově vyvíjenou část
- Naprogramuje se část
- Testové případy se použijí k otestování části (fakultativně), provádí programátor
- Testové případy se integrují s ostatními testy, systém se integruje
- Systém se při agilních postupech integruje a testuje jako celek, to lze u menších a nekritických aplikací či u nepříliš složitých služeb v SOA

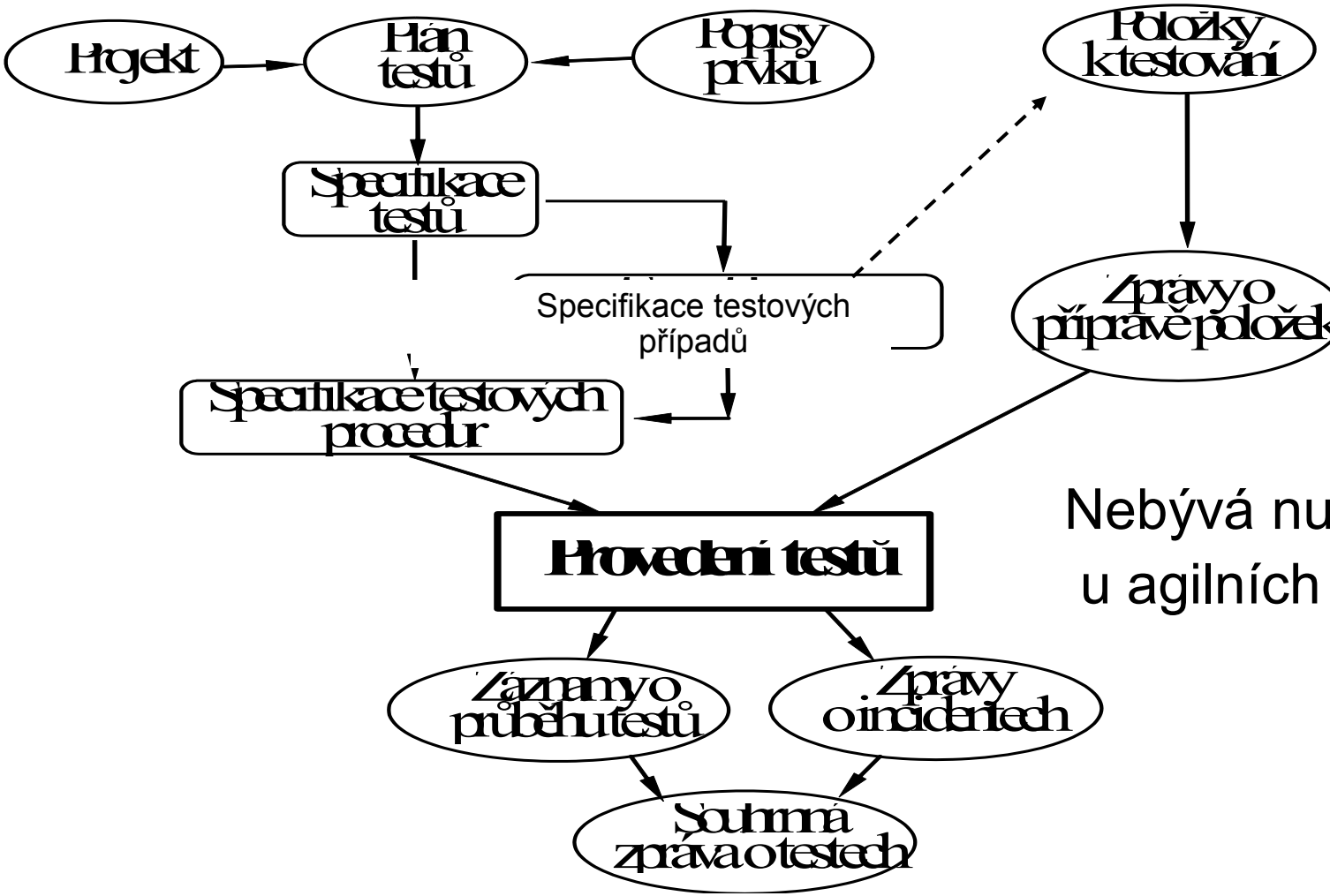
Specifikace testů, neagilní případ, složité či kritické části

- Id
- Podmínky a způsob provedení
- Popis testu, testové procedury
- Kriterium přijetí/zamítnutí testů
- Kriteria pro přerušení testu
- Rizika

Popis problému (selhání)

- Prováděný testový případ, procedura, test („místo“)
- Skutečné výsledky ve srovnání s očekávanými
- Popis anomálie
- Doba
- Pokusy o opakování
- Kdo testoval
- **Nemá být spojováno s návrhy oprav**

Proces testování



Nebývá nutné
u agilních forem

Souhrnná zpráva o testech

- Zprávy o předání položek
- Žurnál testů
- Zprávy o selháních (incidentech)
- Souhrnné hodnocení
  - Co se vše testovalo
  - Hodnocení výsledku (přijmout/nepřijmout testovaný produkt, případná opatření)

Testové metriky (Příklady)

1. Počet modulů modifikovaných při vývoji/změně.
2. Počet defektů odstraněných v dané etapě.
3. Pro modul počet defektů na tisíc řádků.
4. Počet změněných příkazů/míst.
5. Doba na lokalizaci a odstranění chyby.
6. Druhy a frekvence selhání systému.
7. Výčet modulů s největším (nejmenším) počtem defektů.
8. Výčet modulů, které jsou nejsložitější, tj. těch, pro něž nějaká metrika nabývá extrémních hodnot nebo překračuje nějakou hodnotu.

Evidence příčin selhání

- chyba specifikací
- chyba návrhu,
- kódovací chyby,
- selhání hardwaru,
- chyba v reakci softwaru na selhání hardwaru (př. Škoda Plzeň)
- chyba obsluhy

Využití testovacích metrik

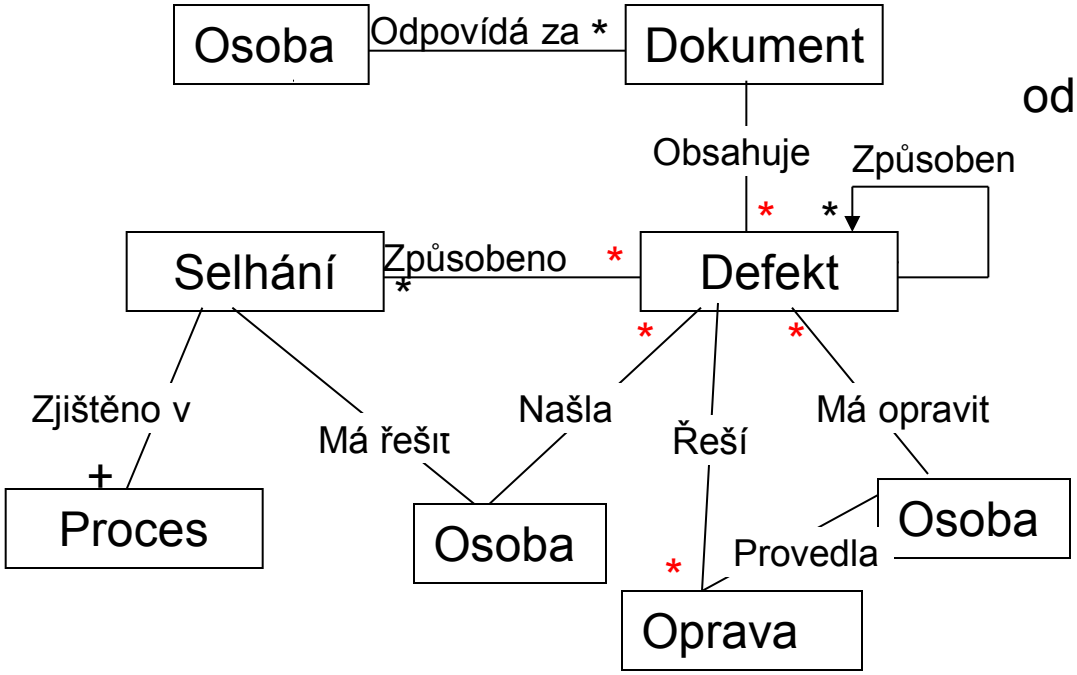
- Kdy ukončit testování
- Dodatečná kontrola efektivnosti oponentur
- Ověřování kvality nástrojů a jejich efektů
- Skrytě hodnocení členů týmu
- Lze použít technologie hodnocení kvality technických výrobků (střední doba mezi poruchami, kritické části)

Datová báze výsledků testů (selhání) by měla být stejná jako u oponentur a u reklamací, *Cíl je identifikovat (možnost) selhání defekty se detekují v následných aktivitách*

Je třeba dohledat prapříčiny

Konceptuální schéma, opakování

- Jednoduchá datová struktura pro vyhledání zdrojů problémů + integrace s oponenturami



V procesu odstraňování selhání se * změní na +

Proces je oponentura, test, nebo stížnost uživatele, technicky odkaz na zápis

Jak se dá zjistit, kdo udělal opomenutí

Permanent obsolence

- Dříve než systém otestuji se změní požadavky a všeobecné podmínky
  - Použité normy zastarají
  - Požadavky se změní
  - Trh se změní, mám jiné obchodní parnery
  - Změní se paradigma
- Řešení
  - Dělám po částech (komponenty, služby)
  - Koupím, znovu použiji, outsourcuji
  - Zlepším své procesy vývoje a údržby
  - **Použiji dokumentově orientované SOA**

Testy dokumentově orientovaném SOA

- Otestují se části ovládáním z obrazovky
- Otestuje je správnost síťové podpory
- Inkrementálně se připojují další služby
- Některé služby se převezmou (trvale nebo dočasně)
- Celý vývoj může být inkrementálně
- Údržba hladce navazuje na vývoj
- **To vše dává možnost oddálit či úpně odvrátit zastarávání. Systém má architekturu podobnou městu (ani bazar, ani katedrála)**

Zkušební provoz

- Po předání je nutná jistou dobu větší péče ze strany dodavatele, viz u-křivku počtů incidentů.
- Předání do běžného provozu je spojeno s dohodami o údržbě, ta v moderních SW architekturách splývá s vývojem.
- U dokumentově orientovaných systémů je žádoucí účast uživatelů resp expertů z ne IT oborů