Motto: [Josef Čapek]
Jistoty jsou dobré, nejistoty přitažlivé,
tajemství vyzývavá

Quo vadis, SI?
aneb
Lesk a bída softwarového inženýrství

Karel Richta
katedra počítačů, FEL ČVUT Praha
richta@fel.cvut.cz

Softwarové inženýrství – co to je, proč to je, co nám to přineslo, co nám to odneslo, kam to směřuje?

- *Příspěvek je podkladem pro diskuzi na téma „softwarové inženýrství“.*
- *Pokouší se vymezit obsah termínu „softwarové inženýrství“.*
- *Pokouší se odhadnout kam softwarové inženýrství směřuje.*

Úvodem trochu historie I.

- Termín „**software**“ zavedl v roce 1958 statistik John Tukey (také autor termínu „bit“).
- Za okamžik zrození termínu „**softwarové inženýrství**“ se obvykle považuje rok 1968, kdy NATO sponzoruje první konferenci s tímto názvem a na toto téma.
- V roce 1969 na ni navázala konference „Techniky softwarového inženýrství“.
- V roce 1972 vychází první časopis „Transactions on Software Engineering“ (IEEE Computer Society).
- V roce 1976 vytváří IEEE Computer Society první komisi, která by měla definovat obsah oboru „softwarové inženýrství“.

Proč to vůbec vzniklo?

- Něco bylo špatně ☹
- Počítačů přibývalo, přibývalo i softwarových projektů, ale ubývalo úspěšně dokončených projektů
- Někdy to došlo až na hranici únosnosti – software byl, nebo mohl být, příčinou havárií (např. Mariner I.)

Úvodem trochu historie II.

- Organizátoři první konference „Softwarové inženýrství“ v roce 1968 zvolili termín „softwarové inženýrství“úmyslně jako provokativní – naznačující, že produkce software musí přejít na jiné postupy a být podložena teoretickými disciplinami, podobně, jako je tomu u inženýrského přístupu v jiných oborech.
- Konala se ve Spolkové republice Německo ve známém středisku Garmisch-Partenkirchen a řídil ji profesor Bauer z Mnichovské techniky.
- Účastnilo se jí asi 50 odborníků z různých oblastí, z praxe i ze škol. Její účastníci formulovali směry, kterými by se výzkum v oboru SI měl ubírat.

Termín softwarové inženýrství

- *„Softwarové inženýrství je disciplina, která se zabývá zavedením a používáním řádných inženýrských principů do tvorby software tak, abychom dosáhli ekonomické tvorby software, který je spolehlivý a pracuje účinně na dostupných výpočetních prostředcích.“*

Konference „Softwarové inženýrství 1968“

Termín softwarové inženýrství

- *„Softwarové inženýrství je disciplina, která se zabývá zavedením a používáním řádných inženýrských principů do tvorby software tak, abychom dosáhli ekonomické tvorby software, který je spolehlivý a pracuje účinně na dostupných výpočetních prostředcích.“*

Konference „Softwarové inženýrství 1968“

Termín softwarové inženýrství

- *„Softwarové inženýrství je disciplina, která se zabývá zavedením a používáním řádných inženýrských principů do tvorby software tak, abychom dosáhli ekonomické tvorby software, který je spolehlivý a pracuje účinně na dostupných výpočetních prostředcích.“*

Konference „Softwarové inženýrství 1968“

Další historie III.

- V roce 1979 vytváří Fletcher Buckley standard IEEE 370 pro vytváření plánů zajišťujících kvalitu software.
- V roce 1986 vzniká standard IEEE 1002, který definuje taxonomii softwarově inženýrských standardů.
- 1990 – 1995 vznikají standardy pro proces životního cyklu software (Standard for Software Life Cycle Processes) - standard ISO/IEC12207, vychází z DoD Std 498.
- V roce 1993 vznikají komise IEEE a ACM, které ústí do společného úsilí definovat softwarové inženýrství jako disciplinu.

Další historie IV.

- V roce 1998 společná komise IEEE a ACM definuje profesi softwarového inženýra.
- Nakonec v roce 2004 vzniká společný návrh „curricula“ pro výuku tohoto oboru, označovaného SE2004.
- Tím se završilo uznání softwarového inženýrství jako discipliny, podobně jako CS1991 završilo uznání informatiky.

Příčina vzniku SI?

- Obvykle se říká, že to způsobil jev nazývaný „**softwarová krize**“.
- Dokud výkon počítačů nepřesáhl určitý rozměr, bylo možno se spolehnout na programátorské „hvězdy“.
- Často se počítače se využívaly pro vědecko-technické výpočty, kde záleželo spíše na preciznosti řešení, než na efektivitě tvorby programů.

Moorův zákon:

- ***„Výkon hardwaru vzrůstá zhruba dvakrát za dva roky“.***
  - Přestože sám autor prohlásil svou extrapolaci jako „pěkně divokou“, zákon zhruba platí dodnes.
  - Firma Intel nedávno zveřejnila výsledky výzkumné zprávy uvádějící, že Moorův zákon pravděpodobně přestane platit až kolem roku 2021 (křemík se dostane na hranici svých možností).

Softwarová krize

Edsger W. Dijkstra:

- *„Hlavní příčinou softwarové krize byl nárůst výkonu hardware. Jinak řečeno, programování nemělo problémy, dokud neexistovaly počítače. Dokud jsme měli slabé počítače, mělo programování jen snesitelně těžké problémy. Nyní máme gigantické počítače a k nim gigantické problémy se softwarem“.*

Příčiny softwarové krize

- Příčinou softwarové krize vždy byl nesoulad mezi složitostí vytvářeného produktu a relativní nedostatečností a nezkušeností softwarové profese. Tento rozdíl způsobuje rozevírání nůžek a důsledkem pak jsou softwarové krize.

- *Pozn. Tento jev asi není omezen na software, ale zdá se mít univerzálnější platnost.*

Projevy softwarové krize

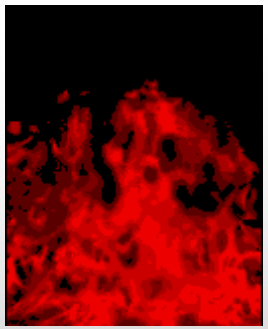
- Projekty překračují rozpočet.
- Projekty překračují čas.
- Software nemá dostatečnou kvalitu.
- Software neodpovídá požadavkům.
- Projekt není dobře řiditelný a software je obtížně udržovatelný.

Skončila softwarová krize?

- Při pohledu na předchozí body a současné projekty se zdá, že softwarová krize trvá stále.
- Svůj vliv zde mají též možnosti softwarových expertů, kteří jakkoli jsou geniální, mohou přímo mentálně obsáhnout jen určitý rozsah problémů.
- Řešení velkých projektů nelze proto nechat pouze na nich. Je nutno použít standardní techniku řešení obtížných problémů – „**rozděl a panuj**“ – velký problém je třeba rozdrobit na problémy menší.

Zkušenost (Brooks):

- *„Přidání nových kapacit na zpožděný projekt prodlouží jeho řešení“.*



Tvorba software ↔ inženýrství

- Všechny tyto problémy související se softwarovou krizí vedly tedy nakonec k pokusu udělat z vývoje produkovaného nadšenci inženýrskou disciplinu.
- V 70-tých letech dochází k formulaci základních principů tohoto oboru.
- Vzniká také první generace nástrojů pro podporu této discipliny, které jsou označovány jako **CASE** (Computer Aided Software Engineering).

Co to je inženýr?

- *Absolvent vysoké školy technického zaměření?*
- *Pro zajímavost definice FEANI (Federace evropských národních asociací inženýrů)*

Etika osobnosti inženýra (FEANI)

Inženýr:

- ***vykonává svou činnost na co nejvyšší úrovni - respektujíc zákony země, v níž působí - tak, aby jím poskytované služby byly v souladu s tím, co je v jeho profesi považováno za úroveň odpovídající současnému stavu poznání,***
- ***zachovává profesionální poctivost a intelektuální čest jako záruku nestrannosti v analýze a úsudku a v následném rozhodování,***
- ***je vázán každou v dobré víře uzavřenou smlouvou, na kterou dobrovolně přistoupil,***
- ***v souvislosti s výkonem své profese nepřijímá žádné peníze bez souhlasu svého zaměstnavatele,***
- ***projevuje svou oddanost inženýrské profesi účastí na činnosti inženýrských organizací, zejména takových, které působí při ochraně profesních zájmů a přispívají k rozšiřování vědeckotechnických poznatků a k trvalému zvyšování odbornosti svých členů,***
- ***používá pouze ty tituly a označení, na něž má právo.***

Profesionální etika inženýra (FEANI)

Inženýr:

- ***může přijímat pouze takové úkoly a pověření, která odpovídají jeho kvalifikaci a oprávnění - při zajišťování činností ležících mimo tyto hranice spolupracuje s příslušnými odborníky,***
- ***odpovídá za organizování a provádění úkolů, jejichž zajišťování převzal,***
- ***zřetelně a úplně specifikuje služby, k jejichž provádění se zavázal,***
- ***při plnění úkolů, jimiž je pověřen, činí veškeré nezbytné kroky k tomu, aby byla zajištěna bezpečnost osob a majetku,***
- ***přijímá odměnu ve výši odpovídající poskytnutým službám a převzaté odpovědnosti,***
- ***pečuje o to, aby každé odměňování, které souvisí s činností, za niž odpovídá, bylo přiměřené poskytnutým službám,***
- ***usiluje o dosažení vysoké kvality technických řešení a přispívá ke zvyšování jejich úrovně,***
- ***pečuje o vytváření zdravého a příjemného pracovního prostředí pro své spolupracovníky.***

Co to je inženýr?

- Ideální prototyp inženýra představuje C.Smith z Verneova románu Tajuplný ostrov. Uměl vše, vyrobit nitroglycerin i postavit loď.
- Takový inženýr všeuměl mohl existovat v 19. století. Dnes suma inženýrských poznatků již značně překračuje kapacitu šedé kůry mozkové jednoho individua.
- Univerzální inženýr je již jen romantickou představou a v našem století by působil spíše jako diletant.

Rozmanité inženýrské profese

- Stavební inženýr
- Chemický inženýr
- Inženýr architekt
- Genetický inženýr
- Softwarový inženýr
- …
- Sociální inženýr?
- …
- Byli stavitelé katedrál inženýři?

Co dělá inženýr?

- Než něco vyrábí, tak si to nejdříve nakreslí, namodeluje.
- K tomu potřebuje zavedenou notaci a vědecky podložené metody a postupy.
- Snaží se používat vhodné technologie tak, aby se dílo vytvořilo spolehlivě, efektivně a aby vydrželo.
- Pojem „**inženýr**“ tedy není přesně definován, chápe se spíše intuitivně.

Co to je softwarové inženýrství?

Wikipedie: Pojem „**softwarové inženýrství**“ není nijak jednotný, může mít víc významů:

- Obecný termín, který znamená mnoho činností, dříve označovaných jako **programování**? – ***rozhodně ne***
- Obecný termín, který znamená praktickou činnost s počítači, narozdíl od teoretického přístupu, který se nazývá **informatika**? – ***rozhodně ne***
- Argument pro jisté přístupy k programování se zaměřením na **inženýrskou profesi**, nikoli jako pohled na programování jako druh umění, řemeslné zručnosti a kultury? – ***částečně ano***
- Softwarové inženýrství je definované jako standard IEEE 610.12.? – ***spíše ano***

Definice IEEE 610.12:

- *„Softwarové inženýrství je aplikace systematického, disciplinovaného, kvantifikovatelného přístupu k vývoji, provozu a údržbě softwaru, tj. aplikace inženýrství na software. Také je to studium přístupů dle výše uvedeného.“*

Je vývoj softwaru umění, věda nebo rutina?

Softwarové inženýrství má blízko k různým disciplínám:

- Na jedné straně je možné jej považovat za inženýrství, neboť se jedná o disciplinované využívání pragmatických zkušeností, tj. rutinní postupy, které se očekávají od inženýra.
- Na druhé straně je možné softwarové inženýrství považovat za vědu, neboť v sobě zahrnuje rozvoj matematických disciplin potřebných k řešení úloh.
- Na straně třetí lze softwarové inženýrství považovat za umění, neboť v sobě zahrnuje aspekty běžně přisuzované umění návrhu – návrh vzhledu, návrh ovládání apod.

Lze SI srovnat s jinými inženýry?

- Srovnáme-li softwarového inženýra s inženýrem stavebním, pak stavební inženýr realizuje stavbu podle modelu, programátor programuje podle modelu.
- Model stavebnímu inženýrovi navrhl architekt (územní rozhodnutí, stavební povolení, realizace stavby), programátorovi softwarový architekt (úvodní studie, návrh architektury, konceptuální model) a návrhář (logický model).
- Chemický inženýr navrhuje postup výroby nějaké látky z ingrediencí, softwarový návrhář navrhuje skladbu celku z rozmanitých komponent.
- Postup přípravy látky v laboratoři je jiný, než postup výroby v továrně, záleží na použité technologii. Softwarový inženýr rovněž využívá různé technologie podle cílového prostředí.

Čím se softwarové inženýrství liší?

- Lze si položit otázku, zda existuji specifické postupy, které odlišují softwarové inženýrství od ostatních inženýrských disciplin.
- Výsledný produkt je nehmotný, distribuuje se jiným způsobem než hmotné produkty.

Jak se SI vyvíjí?

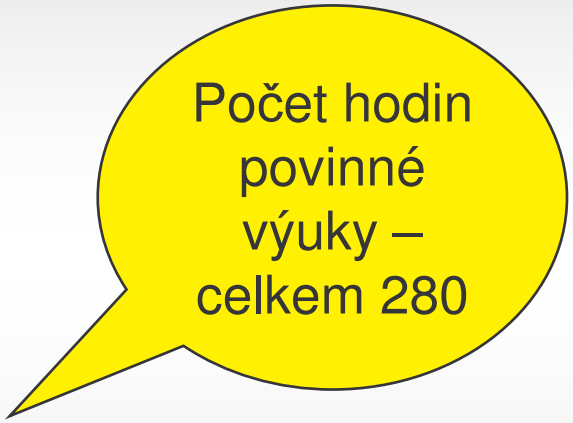
- Softwarové inženýrství bude brzy slavit čtyřicátiny. Zajímavá otázka je, jak se softwarové inženýrství za tuto dobu vyvinulo? Jaké trendy či změny jsou nejpodstatnější? Jiným zajímavým hlediskem je otázka, proč se některé trendy prosazuji s určitým zpožděním, pokud je softwarové inženýrství srovnatelné s ostatními inženýrskými disciplinami.
- Co je v současnosti základní problém, se kterým se softwarové inženýrství potýká? Poučili jsme se již ze zkušeností, podobně jako se to stalo v jiných inženýrských disciplinách?

Sada znalostí softwarového inženýra

- SWEBOK (Software Engineering Body of Knowledge – IEEE 2004)
- SEEK (Software Engineering Education Knowledge - SE2004)
  - Pozn.: Na přípravě se podílejí známá jména jako Pressman, Sommerville, McConnell, ale také Jan Pavelka

Základní členění informatiky (CS)

- Diskrétní struktury (43)
  Základy programování (38)
  Algoritmy a složitost (31)
  Architektura a organizace (36)
  Operační systémy (18)
  Výpočty orientované na síť (15)
  Programovací jazyky (21)
  Styk člověka s počítačem (8)
  Grafika a vizualizace (3)
  Inteligentní systémy (10)
  Správa informací (10)
  Sociální a profesionální otázky (16)
  Softwarové inženýrství (31 – 11%)
  Počítačová věda a numerické metody (0)



Z toho softwarové inženýrství

Povinný základ:

- Návrh
  Programová rozhranní (API)
  Softwarové nástroje a prostředí
  Softwarové procesy
  Požadavky a specifikace
  Validace softwaru
  Vývoj softwaru
  Řízení softwarových projektů

Volitelné doplňky:

- Komponentový vývoj
  Formální metody
  Spolehlivost softwaru
  Vývoj specializovaných systémů

Vedlejší produkt: obory

	Před rokem 1990	Po roce 1990
Hardware	EE + CE	EE + CE
Software	CS	CE + CS + SE
Organizace	IS	IS + IT

EE = elektrické inženýrství (Electrical Engineering)
CE = počítačové inženýrství (Computer Engineering, CE2004)
CS = informatika (Computer Science, CS1991, CS2001)
SE = softwarové inženýrství (SE2004)
IS = informační systémy (IS2002)
IT = informační technologie (IT2006)
Computing Curricula CS1991, CC2001, CC2005

První studijní programy

- Prvý inženýrský program byl vypsán již v roce 1979 na universitě v Seattlu, kde v roce 1982 udělili prvý titul v tomto oboru.
- Prvý bakalářský program v oboru SI vypsalo v roce 1996 vysoké učení technické v Rochesteru. Zpočátku byl odmítnut, ale později akreditaci získal v roce 2003 spolu s inženýrskou školou v Milwaukee.

Základní znalostní oblasti SI

- Správa požadavků (Software requirements)
- Softwarový návrh (Software design)
- Tvorba softwaru (Software construction)
- Testování softwaru (Software testing)
- Údržba softwaru (Software maintenance)
- Správa konfigurací (Software configuration management)
- Řízení vývoje (Software engineering management)
- Softwarový proces (Software engineering process)
- Nástroje a metody softwarového inženýrství (Software engineering tools and methods)
- Kvalita softwaru (Software quality)

Související znalostní oblasti SI

- Počítačové inženýrství (Computer engineering)
- Řízení projektů (Project management)
- Informatika (Computer science)
- Správa kvality (Quality management)
- Manažerství (Management)
- Ergonomie softwaru (Software ergonomics)
- Matematika (Mathematics)
- Systémové inženýrství (Systems engineering)

Co nám SI přineslo?

Řadu novinek, namátkou některé:

- Softwarové profese a týmy.
- Modely životního cyklu.
- Oddělení správy dat od správy funkčnosti.
- Strukturované metody.
- Objektově-orientované metody.
- Komponentové metody.
- Nové programovací jazyky.
- Softwarové zápisy – unifikovaná notace UML.
- Metodiky tvorby softwaru.
- Plánování, metriky, organizace.

Co nám SI odneslo?

Pocit opravdových programátorů:

- Software vytvářejí specializovaní odborníci, kteří jediní znají postupy, nástroje, metody a techniky.
- Plánování a podobné hlouposti sem nepatří.
- Dokumentaci ať si pořizují ti, kteří neumějí číst programy přímo.

Neznámý programátor:

- *„Softwarové inženýrství znamená zavedení discipliny do volné tvorby software. Žádný opravdový programátor ho proto nemůže mít příliš v lásce, neboť jej nutí vytvářet nesmyslnou dokumentaci a další podobné artefakty.“*

Softwarové týmy a softwarové profese

- Jedním z projevů přechodu od ruční výroby k manufaktuře je definice softwarových profesí.
- Řešení velkých projektů vyžaduje spolupráci mnoha řešitelů a práci je nutno rozdělit.
- Dělba práce vyžaduje organizaci týmů řešících větší softwarové projekty.
- Týmy lze organizovat jako strukturované nebo nestrukturované.

Nestrukturované týmy

Dělí práci podle objemu.

Mohou být organizovány jako:

- „Osamělí vlci“
- „Horda“
- „Demokratická skupina“

Strukturované týmy

Dělí práci podle profese.

Mohou být organizovány jako:

- „Chirurgický tým“
- „Tým hlavního programátora“
- „Agilní skupina“
- „Více-týmová organizace“

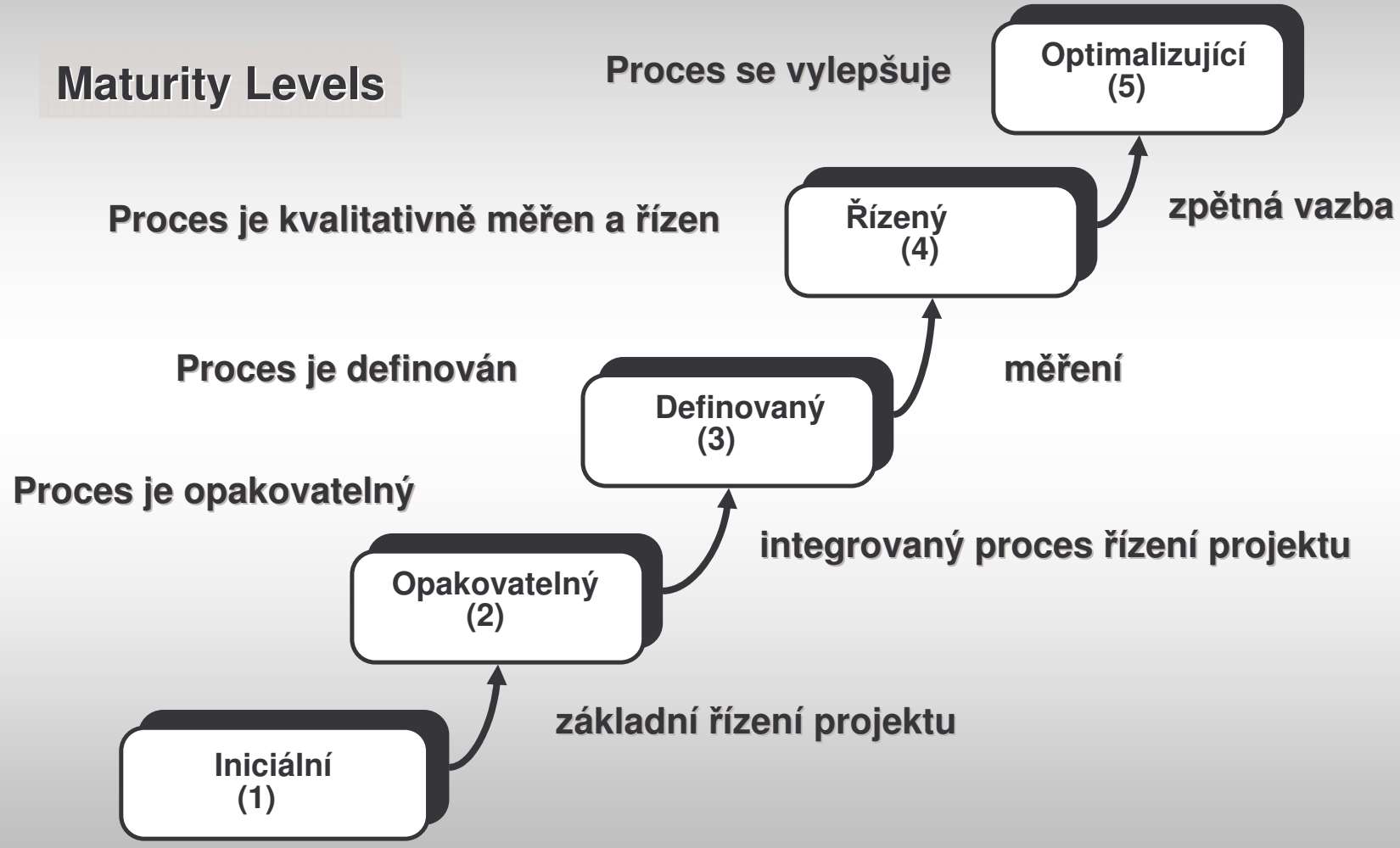
Kterou organizaci zvolit?

- Volba organizace je dána rozsahem projektu.
- Na větší projekty je třeba více-týmová organizace.
- Pro větší projekty se samozřejmě hodí strukturované týmy.
- Máme-li dost prostředků, je nejvýkonnější chirurgický tým.
- Nemáme-li, nejefektivnější může být agilní skupina.

Plánování, metriky, odhady

- Plánování jako nástroj pro snížení rizika
- Notace pro zobrazování plánů – sloupcové grafy (Gantt), síťové grafy (PERT).
- Kritické cesty v plánu, optimalizace.

Úroveň procesu tvorby software



Odhad nákladů na projekt

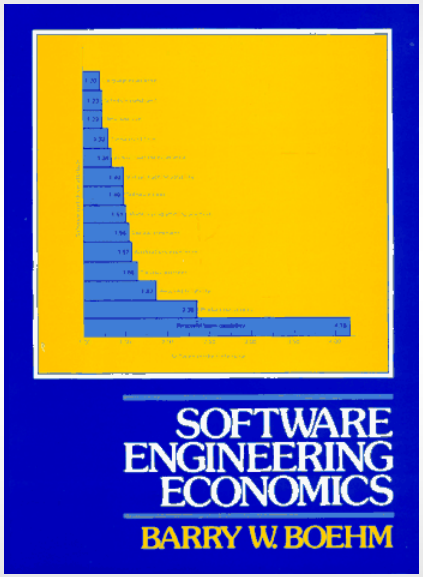
- Odhad na základě zkušenosti z minula
  - již jsme něco podobného řešili a údaje o nákladech jsme si schovali
- Odhad na základě dekompozice problému na odhadnutelné složky (plánování)
  - klasické řešení problému technikou „divide-et-impera“
- Odhady na základě výpočtu z odhadu rozsahu
  - odhad se obvykle řídí odhadem rozsahu kódu (LOC, KLOC, FP)

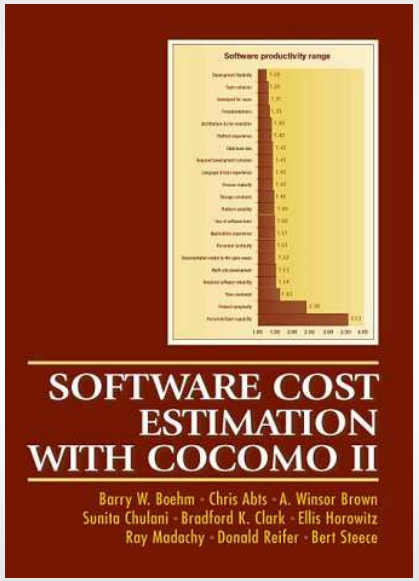
Náklady podle dekomposice na úlohy

- Cena projektu = $\sum$ cen úloh
- Cena úlohy =
  - fixní náklady + cena za použití zdrojů
- Cena za použití zdroje =
  - odměna za práci v normální pracovní době +
  - odměna za práci přesčas +
  - fixní náklady na použití zdroje
- Práce = jednotky * délka
  - Práce je dána součinem počtu jednotek zdroje, které na úloze pracují a délky úlohy

Jiné metody odhadu

- COCOMO (Constructive Cost Model) - Barry Boehm





- http://sunset.usc.edu/research/COCOMOII/

Odhad rozpočtu dle COCOMO

- Vstup: Rozsah produktu v KLOC (KLines of Code)
- Náročnost = 2.94 * (Rozsah) $^{0.91}$
  - (udává se v člověko-měsících)
- Čas = 3.97 * (Náročnost) $^{0.28}$
- Cena = Čas * Plat
- Koeficienty se mění dle typu projektu a korekcí (cca 0.5 ÷ 2.0)

Příklad: Materiální zásobování

- Velikost: 32 KLOC (KDSI)
- Náročnost (Effort): 121.77 ČM
- Čas: 15.50 měsíců
- Lidí: 7.856

(organic mode – jednodušší známé projekty, spočteno přes COCOMO kalkulátor)

http://sunset.usc.edu/research/COCOMOII/cocomo81_pgm/cocomo81.html

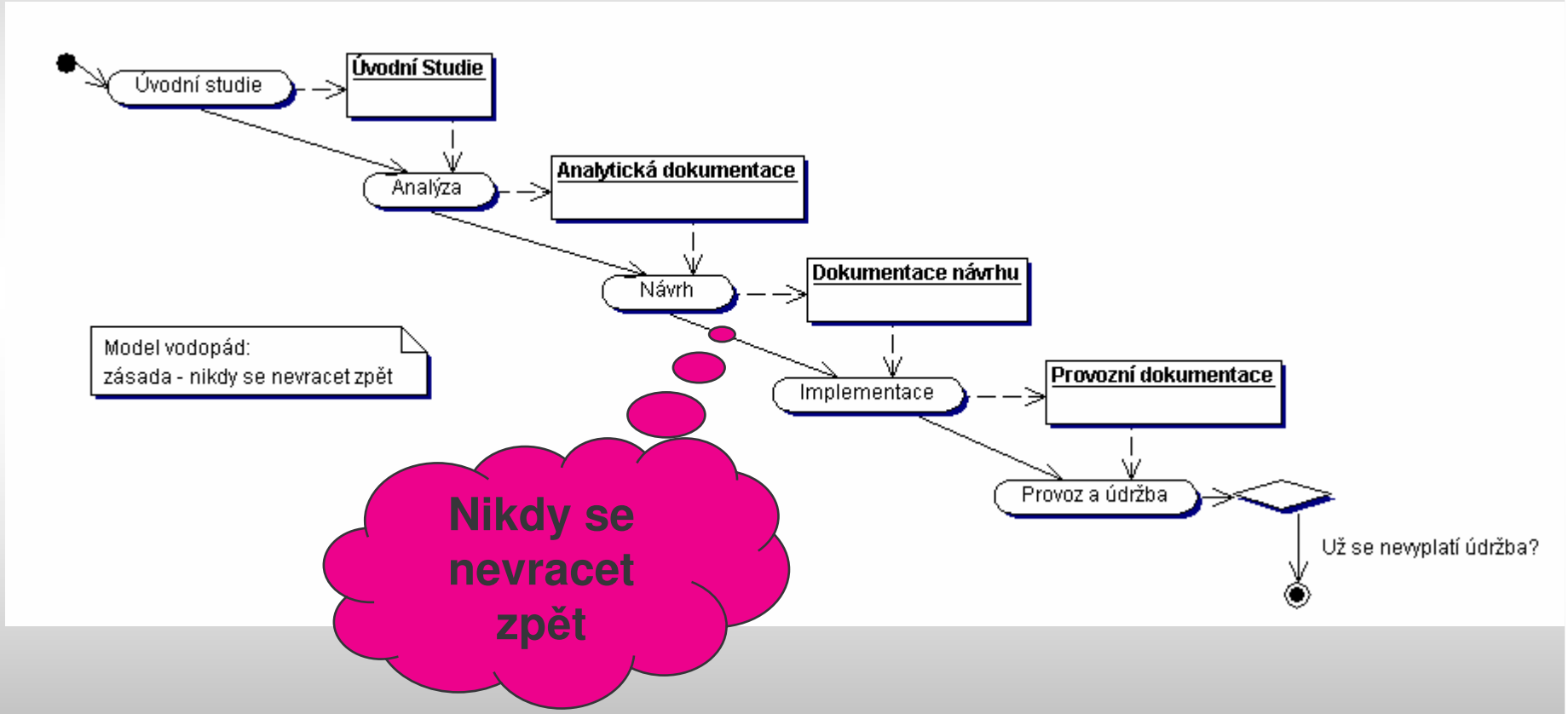
Karnerova metoda odhadu

- Jiná metoda odhadu nákladů, založená na modelu jednání.
- Spočítejte aktéry, každého aktéra zařaďte do kategorie, sečtěte váhy všech aktérů.
- Rozdělte případy použití do kategorií podle odhadu počtu potřebných transakcí, sečtěte váhy všech případů užití.
- Sečtěte obě váhy a získáte neupravenou váhu modelu jednání
- Adjustujte takto spočtenou váhu technickými faktory a faktory prostředí. Získáte tak upravenou váhu modelu jednání.
- Vynásobte UCP předpokládanou pracností jednoho případu užití (cca 15 – 30 hod, Karner doporučuje 20 hod). Získáte pracnost v člověko-hodinách.

Modely životního cyklu

- Vodopádový model
- Model „průzkumník“
- Spirálový model
- Iterativní vývoj
- Přírůstkový model

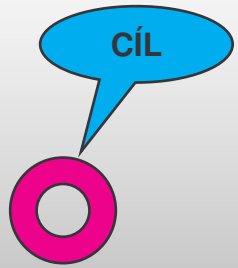
Model vodopád

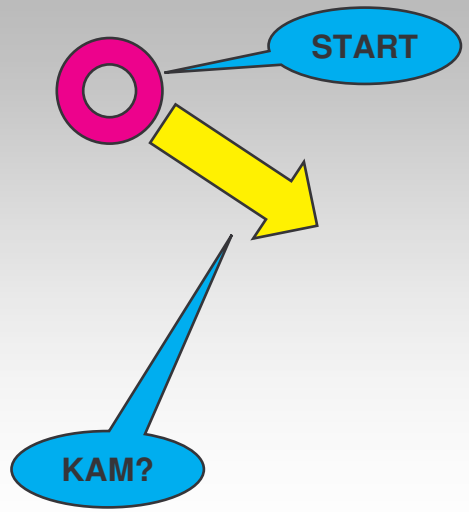


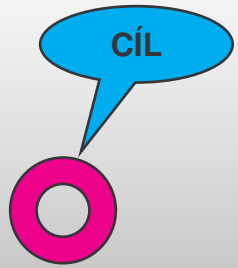

START

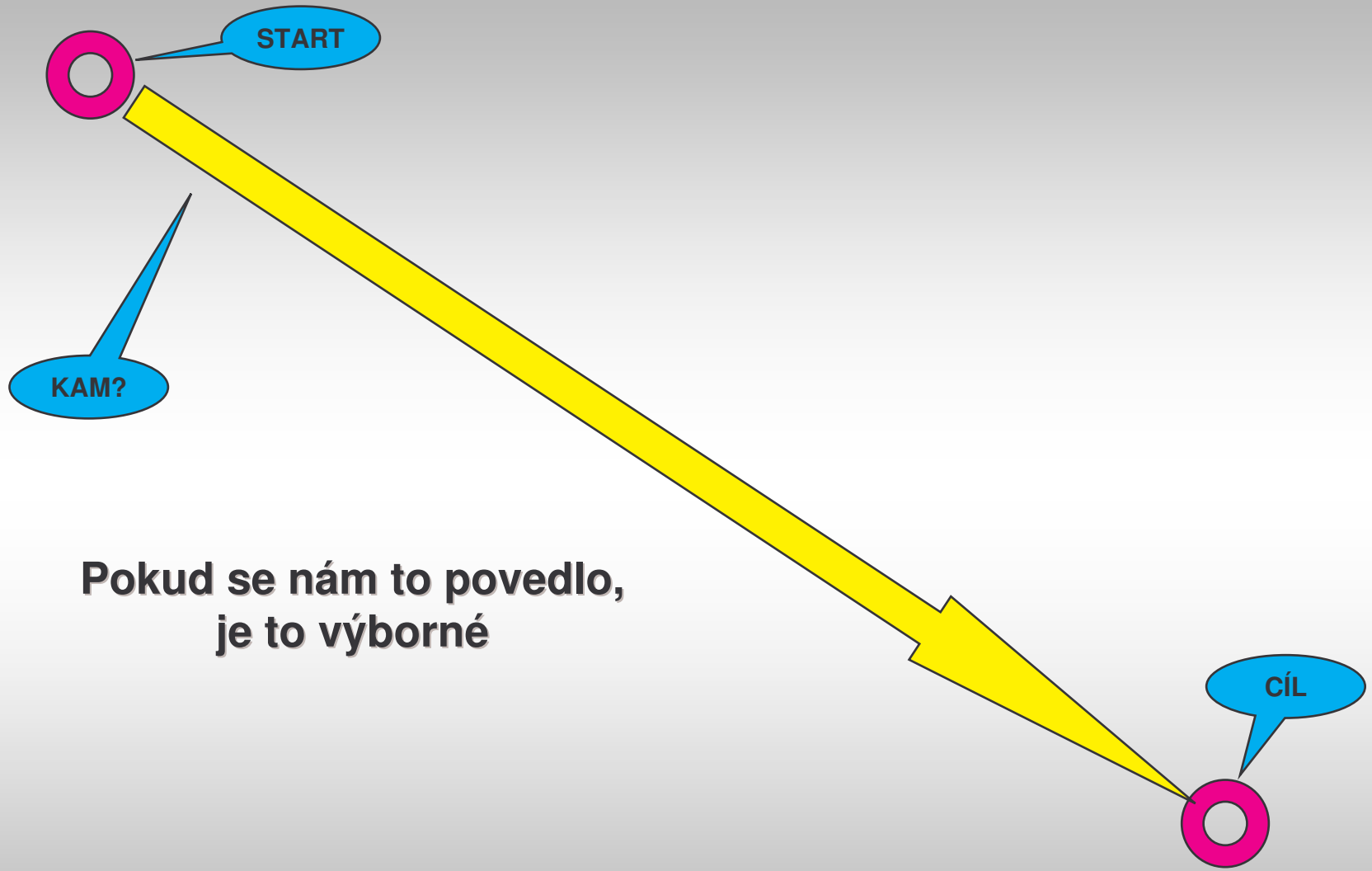

CÍL

Snažíme se specifikovat,
co zákazník chce

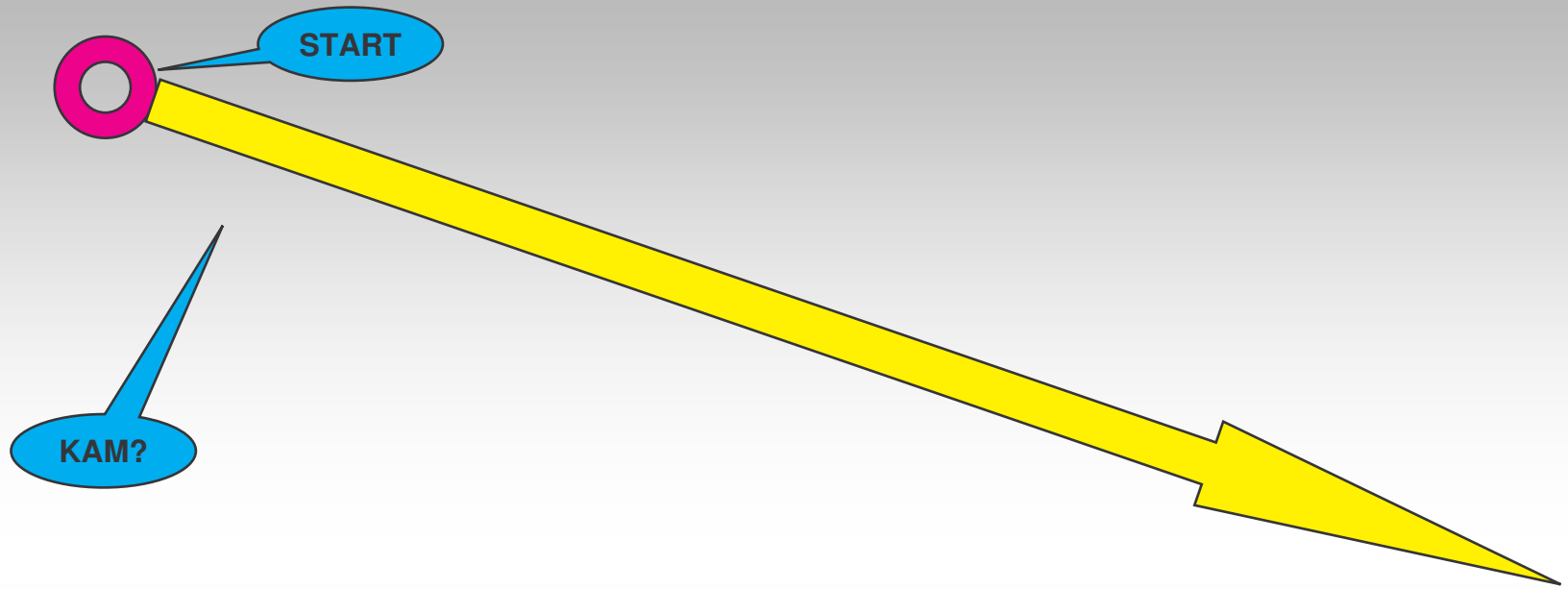



START
KAM?

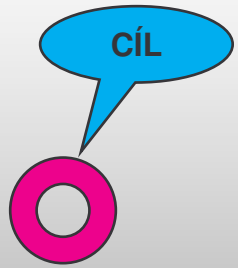

CÍL

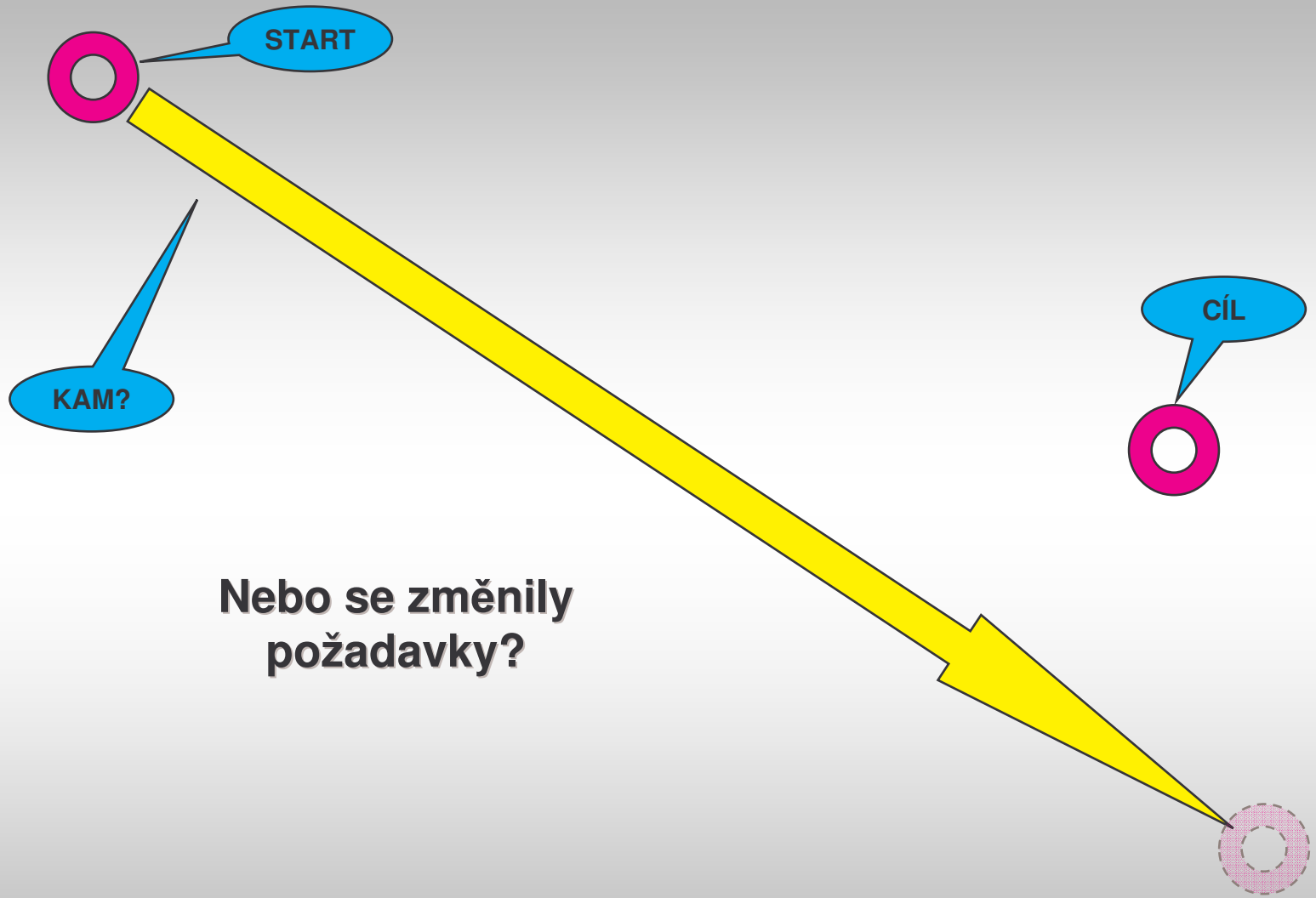


Pokud se nám to povedlo,
je to výborné

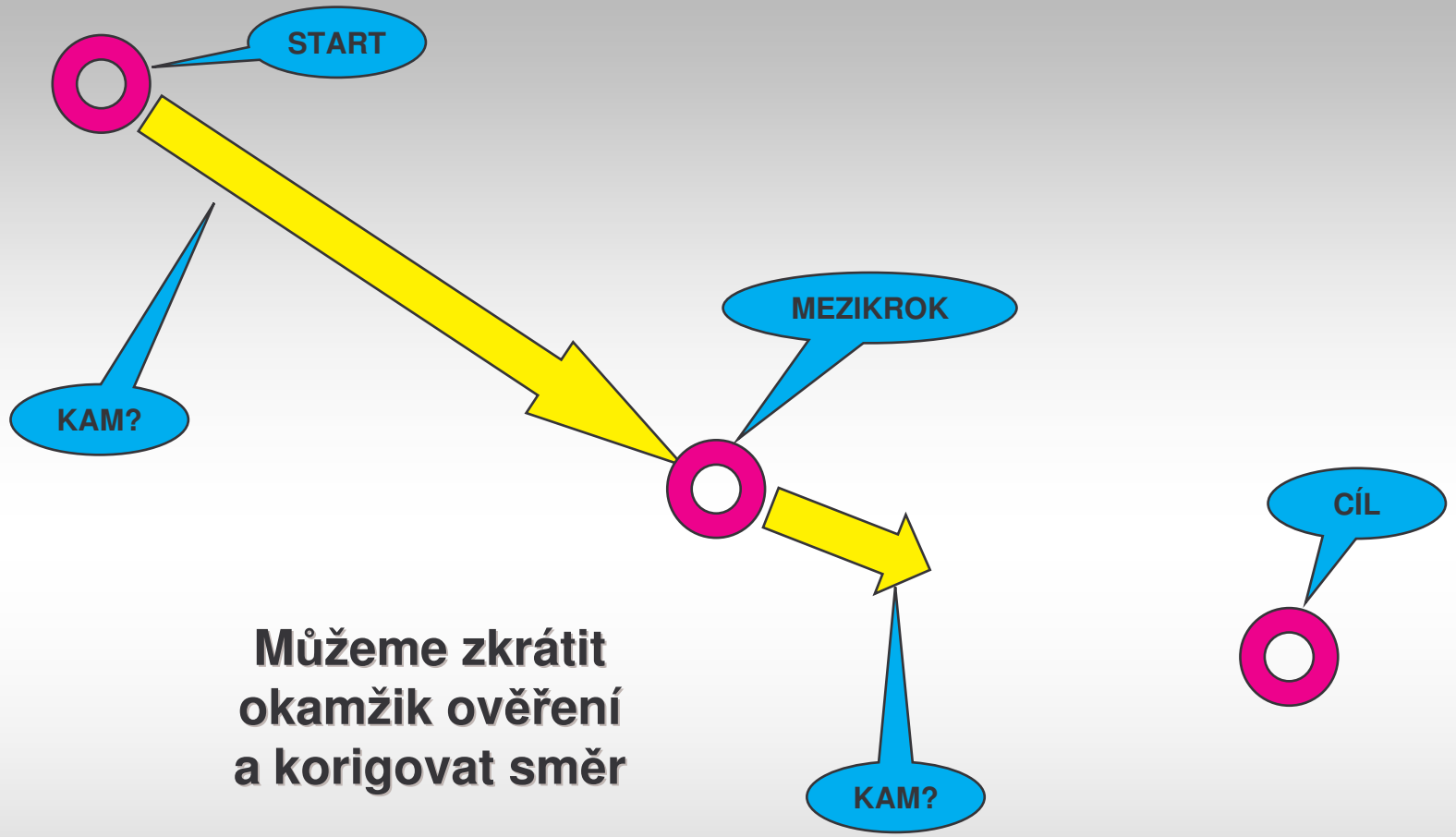


Ale co když jsme se trochu zmýlili?



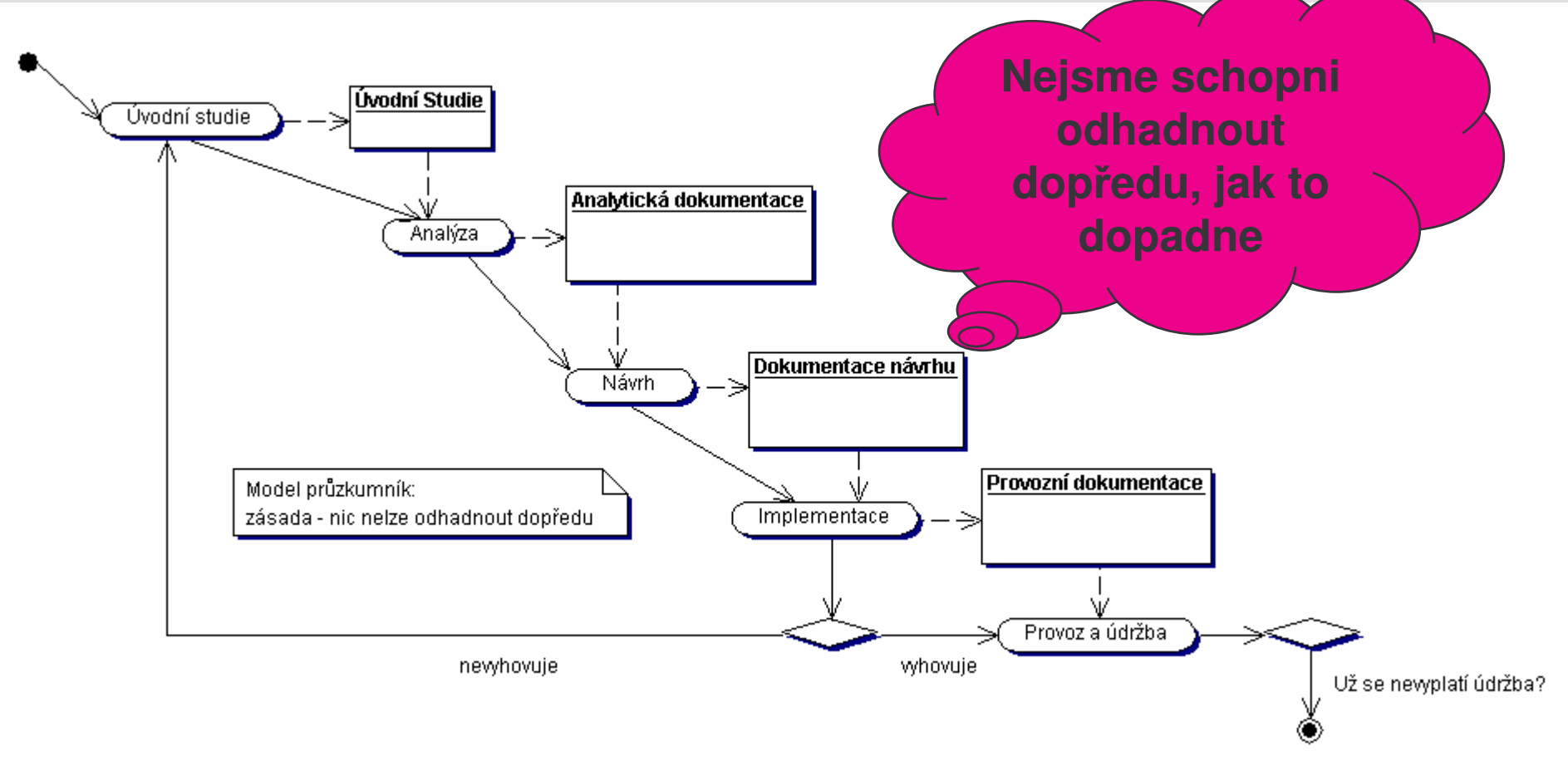


Nebo se změnily požadavky?

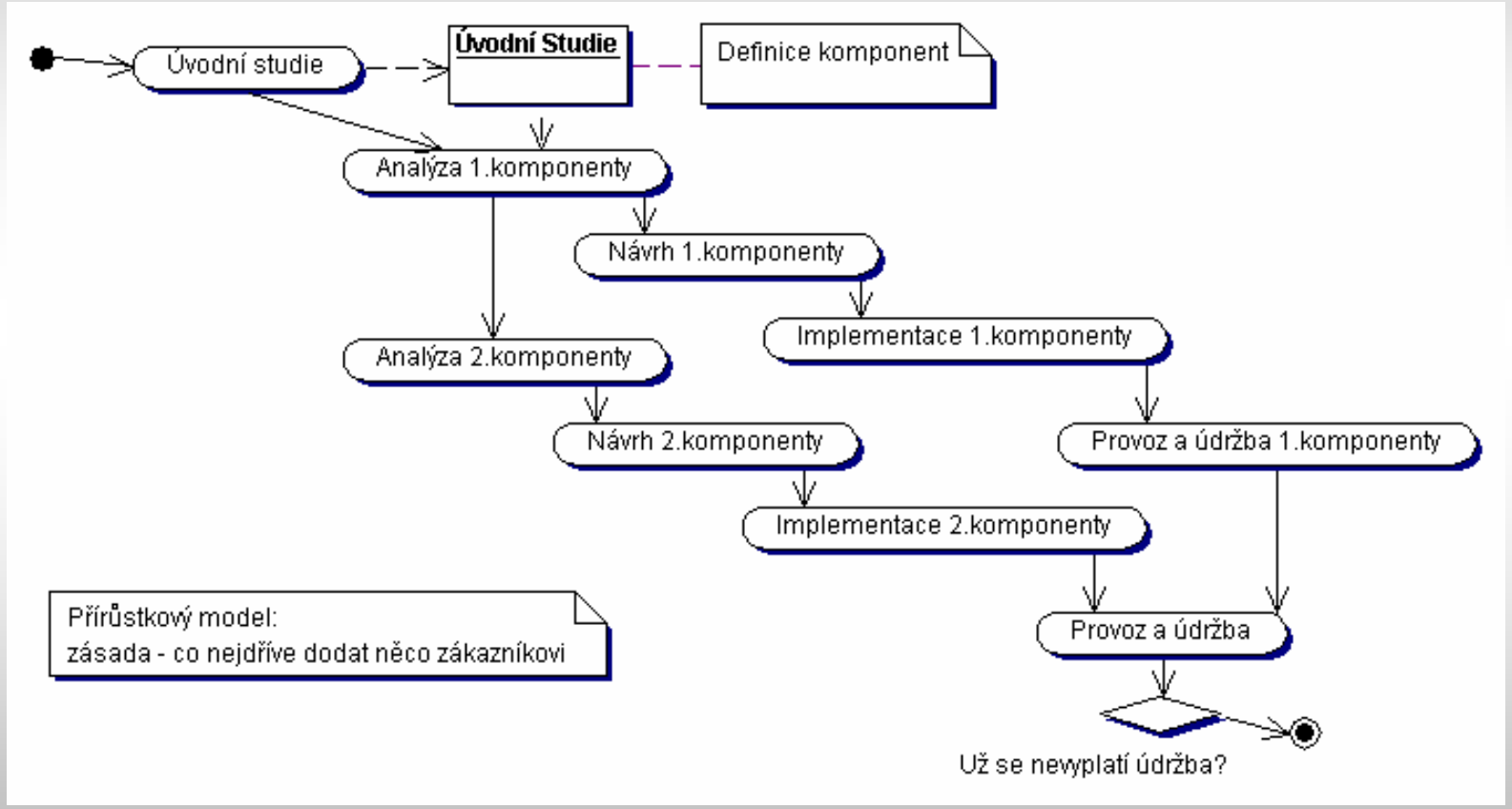


Můžeme zkrátit okamžik ověření a korigovat směr

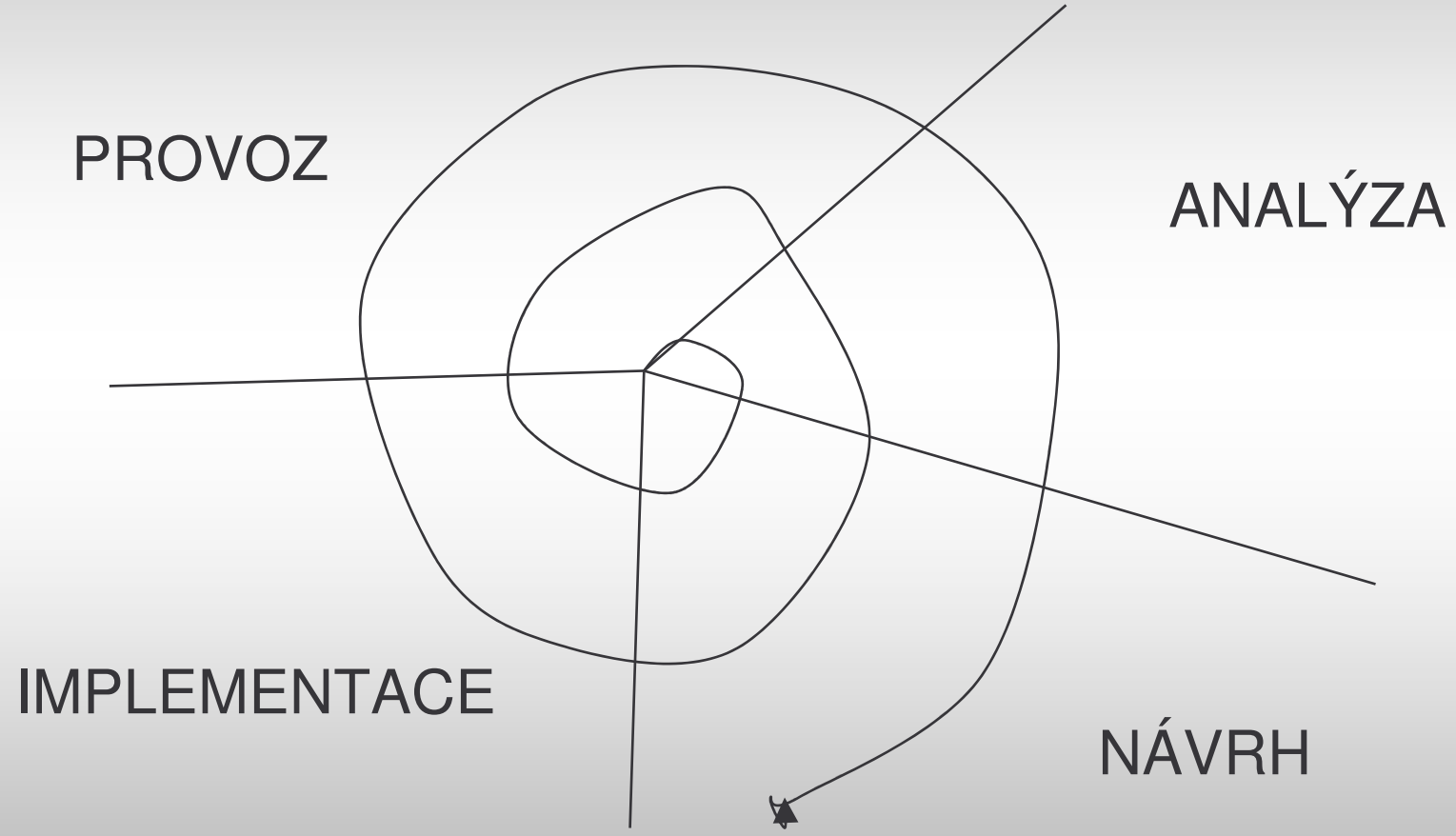
Model průzkumník



Přírůstkový model



Spirálový model



Jiné modely životního cyklu

- Model RAD (Rapid Application Development)
  - model určený pro dobře srozumitelné a dobře vymezené problémy, s malými riziky, využívající krátký vývojový cyklus (cca do 3 měsíců), problém je rozdělen na samostatné moduly
- Evoluční model
  - využívá skládání komponent, které mohou být vyvíjeny současně, či zakoupeny a upraveny
- Formální metody
  - využívají specifikací řízený styl vývoje, tj. generování programů ze specifikací
- Extrémní programování a podobné techniky

Oddělení programů od dat

- Jedním z důležitých důsledků snahy řešit softwarovou krizi a formulace požadavků na softwarové inženýrství je princip oddělení správy dat od programů, které s nimi pracují.
- Opět se jedná o využití techniky „rozděl a panuj“ – správa dat má na starosti bezpečné uložení dat s minimálním rizikem jejich ztráty. Programy operující nad databází se naopak zaměřují na aplikační logiku.
- Databázové systémy představují odvětví softwarového inženýrství, které vzniká opět konce 60-tých let, doby vzniku prvých hierarchických databází.

Databázové systémy

- Prvá úložiště dat jsou organizována hierarchicky.
- Jejich nevýhody se později pokoušejí odstranit databáze síťové.
- Prostou záměnou ukazatelů (adres) klíčovými položkami dospíváme k relačním databázím. Článek pana Codda pochází opět z roku 1970, byť prvé produkční implementace jsou k dispozici až kolem roku 1980.
- Objektové-databáze se vracejí k ukazatelům.
- XML databáze se vracejí k hierarchické struktuře.

Opakované použití a vzory

- Opakované použití řešení jednou vyřešeného problému přináší na jedné straně nesporně mnohem vyšší efektivitu. Má ale svá úskalí v tom, že opakované užití předpokládá nejen správné vyřešení vzorového případu, ale i jeho precizní dokumentaci – vytváření vzorů.

Různé úrovně vzorů

- Programovací vzory, idiomy (ADT, programovací techniky)
- Návrhové vzory (proxy, iterátor, builder)
- Architektonické vzory (klient-server, MVC, PAC)
- Analytické vzory (podvojné účetnictví, bankovní moduly, …).

F. Brooks:

- *„Geniální návrh vytvářejí geniální návrháři. Jakákoliv geniální metodika může podpořit a osvobodit tvořivou mysl, nemůže osvítit nebo inspirovat nádeníka. Rozdíl lze přirovnat k rozdílu mezi Mozartem a Salierim.”*

Quo vadis, SI?

- Další vývoj v oboru softwarového inženýrství závisí na více faktorech.
- Jedním z nich je zatím stálý růst výkonu hardware, se kterým lze počítat ještě pro příští dekádu.
- Pak se ještě může objevit jiná technologie, než je křemík (např. biologické procesory), nebo jiné paradigma (např. kvantové počítače).
- Takový obrat přinese do produkce software jistě potřebu nových konceptů.

A co když se neobjeví?

- Pokud se nová technologie neobjeví, bude opět nutno přejít od expanzivních metod k metodám intenzivním – zvyšování schopností bude zapotřebí dosáhnout zlepšením struktury, kódu apod.
- Současný software v řadě případů plýtvá pamětí, či časem, neboť tyto potřeby zaštítí výkonnější hardware.
- Pokud se vývoj hardware pozastaví, bude třeba sáhnout do rezerv.
- Kdyby se výkon hardware naopak dramaticky zvýšil, bude možno opět uvažovat o nových aplikacích a metodách, což by opět rozevíralo nůžky softwarové krize.

Čekají nás další softwarové krize?

- Softwarová krize konce 60-tých let byla způsobena několika faktory – výkonnost hardwaru přerostla schopnosti programátorů té doby. Používané techniky a metody nestačily novým potřebám.
- Pravděpodobně lze na situaci aplikovat Parkinsonův zákon:

„Každý člověk časem zastává v organizaci místo, na které nestačí.“

- Člověk nastoupí do organizace a zastává určité místo. Pokud se osvědčí, znamená to, že na toto místo stačí a je povýšen na místo vyšší. Takto se postupně dostane na místo, na které nestačí a již není dále povyšován.
- Podobně se každá metodika tvorby softwaru postupně s vývojem možností dostane do situace, kdy se již nehodí.

Neznámý manažer:

„Kdo je na trhu první, obvykle vyhrává.“

- Tento princip aplikuje řada firem, včetně těch největších. Používá se i na situace, kdy se její použití tak úplně nehodí, uživatelé pak fungují jako beta-testéři.

Důsledek:

- Softwarové krize nás tedy jistě ještě čekají - pokud neprobíhá softwarová krize stále.
- V rozsahu katastrof roku 1968 softwarové krize asi již nepřijdou. Softwarový svět se trochu poučil, snaží se vyvíjet nástroje na obranu před takovými jevy.
- Pokud by se měla použít paralela, kapitalistický svět se také poučil z velké krize v roce 1933, ale to neznamená, že v současnosti se žádné krize vyskytnout nemohou a také se vyskytují.
- Jsme na rozdíl od 60-tých let lépe připraveni?
- Jisté indicie naznačují, že nikoliv, nebo alespoň ne všichni.

Možný zdroj další krize

- Za případný možný zdroj další krize je do jisté míry možno považovat tvorbu internetových aplikací.
- Existující jazyky a nástroje umožňující vytvářet programy velkému okruhu lidí.
- Požadavky se nespecifikují, neboť často budoucí uživatele systému ani úplně neznáme.
- Snažíme se ale ovládnout trh, tak raději předhodíme uživatelům cosi jednoduššího, co už ale funguje.
- Postupně pak budeme aplikaci doplňovat o další možnosti.
- Problém začne, když uživatelé začnou od nových aplikací vyžadovat složitější a komplexnější služby. Pak pravděpodobně opět přijdou na řadu specialisté a potřeba sofistikovanějších aplikací.

Jak s těmito příčinami bojovat? I.

Zopakujme si jaké problémy stály za softwarovou krizí:

- Byla to jednak špatná komunikace mezi zúčastněnými na všech úrovních (zákazník, analytik, návrhář, programátor, testér, manažer). Dnes je nutnost a význam komunikace znám.
- Jiný zdroj potíží byl nesprávný přístup k vývoji, kdy se vývojář snažil vytvořit „umělecké“ dílo, nezáleželo mu tolik na spokojenosti zákazníků. Dnes jsou v řadě případů zákazníci zatahováni do procesu vývoje a jejich spokojenost je primární (někdy možná až příliš).
- Také dělba práce je mnohem lépe propracovaná, členové týmů zastávají různé role.

Jak s těmito příčinami bojovat? II.

- Nové technologie většinou přinášejí nové možnosti, je proto třeba je sledovat a studovat. Nelze si ale představovat, že za nás vyřeší všechny problémy a potíže. Vhodná volba technologie je pouze součástí řešení, které může případně usnadnit.
- Dalším problémem bylo nesprávné plánování a špatné odhady. Tyto nedostatky se v současnosti snažíme odstranit propracovanými postupy a metodami.
- Přesto jsou asi správné odhady stále trochu magické a hodně záleží na zkušenosti a umu. Důležité je včasné docenění hrozeb a rizik, neboť jejich podcenění se později nevyplatí. Některé metodiky se proto nechávají rizikem řídit – snažíme se nejprve odstranit a vyřešit nejvíce rizikové části.

Nová paradigmata vývoje SW

- Vytvářej software tak obratně, aby se prodával v co největších kvantech.
- Vysoká kvalita a spolehlivost software vedou k nižší ceně, de-facto standardům, zvyšují přidanou hodnotu.
- Vyráběj z prefabrikátů formou „Vyber a kombinuj“.
- Vytvářej produkty kolaborativně, využívej globální kooperace a konkurence.

Požadavek I.

- Musíme být schopni vytvářet softwarové komponenty, jejichž chování jsme schopni specifikovat a pochopit, a které tedy budeme schopni využívat v jiných komponentách a aplikacích.

- *Jako příklad lze uvést např. komponentu zajišťující provedení platby.*

Požadavek II.

- Musíme být schopni prostřednictvím software poskytovat služby, kde samozřejmě důležitá je nejen poskytovaná funkčnost, ale i další nefunkční charakteristiky – rychlost provedení, doba odezvy apod. Celkového chování služby dosahujeme kooperací použitých komponent, služba tedy může být zprostředkovaná.

- *Takovou službou může být provedení platby prostřednictvím komponenty zajišťující provedení platby.*

Požadavek III.

- Musíme být schopni tyto služby vytvářet takovým způsobem, aby byly připravené na přicházející změny a byly schopny se těmto změnám přizpůsobit, přičemž adaptaci bude možno jednoduše spravovat.

- *Např. pokud přibude nový způsob platby, komponenta pro zajištění plateb by měla být schopna integrovat tento nový způsob na základě jednoduchého pokynu.*

Požadavek IV.

- Musíme být schopni podporovat nové struktury a schémata, nové metody pro rozmanité nové aspekty ve vývoji softwaru.

Požadavek V.

- Musíme být schopni přizpůsobit dobré vlastnosti existujících metod a nástrojů pro nová prostředí, nové agilní, příp. i extrémní metodiky a vůbec neklasické použití, neboť je o čas.

Jaké zbraně máme k dispozici I.:

- Porozumění současnému stavu přináší lepší znalost. Kvalita služby je vždy ovlivněna znalostí, která za službou stojí. Je třeba využívat služeb, za kterými stojí zkušenost.
- Životní cyklus softwarových systémů se zkracuje. Potřebujeme technologie, které podporují postupný vývoj, přidávání dalších komponent a přídavných modulů.
- Měli bychom vytvářet technologie a metodiky, které jsou schopny pracovat s „černými skříňkami“, případně s nespolehlivými komponentami.
- Zařídit administrativní prostředí, které podporuje inovace obchodní i technické.

Jaké zbraně máme k dispozici II.:

- Podporujme typy lidí, kteří jsou schopni absorbovat tyto aktivity.
- Podporujme výukové systémy, ve kterých se propaguje kooperativnost, distribuované kognitivní myšlení.
- Vývoj softwarových metodik pro vývoj softwaru evolucí. Metodiky orientované na určitou doménu, orientované na architekturu.
- Využívání softwarových vzorů, zejména analytických vzorů, návrhových vzorů a idiomů. Prohlubujme znalosti objektově-orientovaného a funkcionálního programování.
- Vytvářejme prostředí pro kooperativní distribuovaný vývoj v počítačové síti.
- Podpora školení pro lidí z průmyslu tak, aby pochopili, že vytvářený software je má podporovat v jejich běžné práci, že není účelem, ale prostředkem.

Odhadované změny v profesi

Podle analytiků z Gartner Group lze v následujícím desetiletí předpokládat změny v profesi IT:

- Bude vznikat potřeba nového druhu IT profesionálů, kteří budou ovládat nejen technologie, ale zejména budou rozumět podporovaným procesům.
- Nové cíle IT profesionálů lze formulovat asi takto: „Strávil jsem dva roky tím, že jsem pomáhal navrhnout a realizovat proces prodeje přes internet, což přineslo zvýšení obratu o 20%“.
- V roce 2010 to postihne šest z deseti IT profesionálů, velké firmy budou redukovat IT.

Globální zdroje

- Díky vysokorychlostním globálním sítím s možností vyhledávání znalostí a služeb, stanou se globální zdroje standardní součástí portfolia a vzniká tím pro mnoho IT profesionálů konkurence.

Automatizace

- Automatizace v oblasti produkce a nasazování software rovněž změní produkci a vývoj softwaru a ovlivní související profese. Např. automatizace testování, monitorování vzdálených systémů, zajištění technické podpory apod.
- Sem také patří modelem řízený vývoj (MDD) a architektura (MDA)

Konzumní využívání služeb

- Mnohem větší význam bude mít konzumní využívání služeb. Prostřednictvím technologií jako jsou osobní počítače, dostupné služby, mobilní telefony apod. budou služby požadovány větším okruhem méně znalých lidí.

Restrukturalizace obchodu

- Restrukturalizace obchodu – slučování firem, akvizice, odhalování rezerv, konsolidace, řešení nezaměstnanosti, využívání vnějších zdrojů, bankroty firem – to vše by mělo přivést IT profesionály k otázce, zda mají zůstat u „čisté technologie“, nebo se pokoušet rozšířit znalosti do oblasti průmyslu, obchodu, porozumět základním procesům, což jim přinese výhodu všestrannosti v této doméně.

Primární oblasti budoucího využívání softwarových technologií

- Rozvoj infrastruktury a služeb (hardware, sítě, služby s tím související).
- Vzrůstající požadavky na „obchodní inteligenci“ (business intelligence), poskytované on-line služby.
- Návrh procesního zpracování, vylepšování obchodních a technických procesů, případně jejich automatizace.
- Správa distribuovaných vztahů a zdrojů.

Závěr

- Softwarové inženýrství jako disciplina vzniká v 60-tých letech, kdy se poprvé projevilo zpoždění vývoje softwaru za vývojem možností hardwaru. Důsledkem byla softwarová krize a reakcí na ni snaha zavést do tvorby software inženýrské postupy. Inženýrským přístupem se obecně myslí opodstatněné využívání nových vědecko-výzkumných poznatků, disciplinovaný postup při řešení konstruktérských problémů.

Albert Einstein:

- ***„Snažte se věci udělat tak jednoduché, jak to jde, ale ne jednodušší.“***

The End