

# Úvod do softwarového inženýrství

## IUS 2024/2025

### 1. přednáška

Ing. Radek Kočí, Ph.D.  
Ing. Bohuslav Křena, Ph.D.

16. a 20. září 2024

# Organizace předmětu – Přednášky (1/2)

D105 + D0206  
**1BIA + 2BIA + 2BIB**

**pondělí 16:00 – 18:50**



Ing. Radek Kočí, Ph.D.

D105 + D0206 + D0207  
**1BIB + 2BIA + 2BIB**  
**+ FP**

**pátek 13:00 – 15:50**



Ing. Bohuslav Křena, Ph.D.

# Organizace předmětu – Přednášky (2/2)

- V **pondělí 28. října 2024** přednáška odpadne (státní svátek).
- Jedna páteční přednáška odpadne také (stejný počet přednášek).
- Posledních 10 minut je vyhrazeno pro studijní koutek.
- Konzultace: o přestávce, po přednášce, diskuzní fóra, osobně, e-mail
- Předmět e-mailu necht' začíná textem *[IUS]*.  
Zvýšíte tím šanci na jeho vyřízení.
- Děkujeme prof. M. Bielikové za poskytnutí původních přednášek.

# Organizace předmětu – Cvičení (1/2)

## Témata

1. Specifikace požadavků v UML – diagramy případů užití, diagramy aktivit a stavové diagramy (3. a 4. týden výuky)
2. Datové modelování – ER diagramy (5. a 6. týden výuky)
3. Diagramy tříd a diagramy objektů (7. až 9. týden výuky)
4. Sekvenční diagramy a diagramy komunikace (9. až 11. týden výuky)

## Organizace

- Student absolvuje 4 dvouhodinová cvičení za celý semestr.
- Na cvičení je potřeba se přihlásit ve STUDISu.

## Hodnocení

- Za aktivní účast lze na každém cvičení získat **3 body**.
- Chyby ani neznalosti ke ztrátám bodů za cvičení nevedou.
- Nečinnost a nezapojení se do cvičení však ano.

# Organizace předmětu – Cvičení (2/2)

- **Nemoc** či jinou překážku ve studiu lze řešit se cvičícím
  - účastí na jiném cvičení nebo
  - při ohlášení na Studijní oddělení přidělením bodů podle
    - domácí úlohy (1. a 2. cvičení)
    - týmového projektu (3. a 4. cvičení)

## Asistenti pro cvičení, domácí úlohu i pro projekt

- Maksim Aparovich (in English)
- Ing. David Chocholatý
- doc. Vladimír Janoušek
- Ing. Samuel Olekšák
- doc. Adam Rogalewicz – vedoucí, FP
- Ing. Michal Rozsival
- Ing. Michal Šedý
- Ing. Pavol Vargovčík
- Ing. Petr Veigend

# Organizace předmětu – Domácí úloha

## Téma: ER diagram (12 bodů)

- ER diagram (Entity Relationship Diagram) – uchovávaná data
  - Probírá se na 3. přednášce a na 2. cvičení.
  - Obdobný diagram čkejte na zkoušce.
- Přihlašování na variantu zadání a odevzdání v Moodle.
- Přihlašování na variantu zadání od **7. října 2024, 20:24**.
- Odevzdání do **3. listopadu 2024, 23:59**.
- Konzultace s asistentem, který danou variantu zadal.

**Domácí úlohu vypracovávejte samostatně (bez AI)!**

# Organizace předmětu – Týmový projekt

## Téma: Komplexní model informačního systému (16 bodů)

- ER diagram + probírané UML diagramy
- Tým může mít 4 nebo 5 členů.
- Možnost přerozdělení bodů v rámci týmu (méně aktivní)
- Přihlásíte se na jednu z cca 40 variant zadání v Moodle.
- Konzultujte s asistentem, který danou variantu zadal.  
Řešení ani konzultace nenechávejte na poslední chvíli.
- Zahájení ... v pondělí **4. listopadu 2024, 20:24**
- Odevzdání ... do neděle **1. prosince 2024, 23:59**
- Obhajoba ... 12. a 13. týden výuky
- Prezentující bude z týmu vylosován.

**Projekt vypracovávejte pouze v rámci svého týmu (bez AI)!**

# Organizace předmětu – Zkouška

- Studium je investice do vzdělání, která má vysokou návratnost.  
lepší pozice s vyšším platem
- Je především na Vás, jak bude investice 3-5 let života úspěšná. FIT nabízí v praxi vysoce ceněné vzdělání, ale nemůže ho studentům vnutit proti jejich vůli. Snaží se ale chránit své dobré jméno a atraktivitu svých absolventů na trhu práce.
- *Zkouškou se zjišťuje komplexní zvládnutí látky vymezené v dokumentaci předmětu prezentované ve výuce na úrovni odpovídající absolvované části studia a schopnosti získané poznatky samostatně a tvůrčím způsobem aplikovat. (SZŘ VUT čl. 13 odst. 2)*
- **Pro získání bodů ze zkoušky je nutné zkoušku vypracovat tak, aby byla hodnocena nejméně 30 body (ze 60). V opačném případě bude zkouška hodnocena 0 body. To platí i pro studenty FP.**
- Nezkoumejte, jak projít studiem s co nejmenším úsilím, ale sami se snažte naučit co nejvíce. Ovlivní to celou Vaši profesní kariéru!

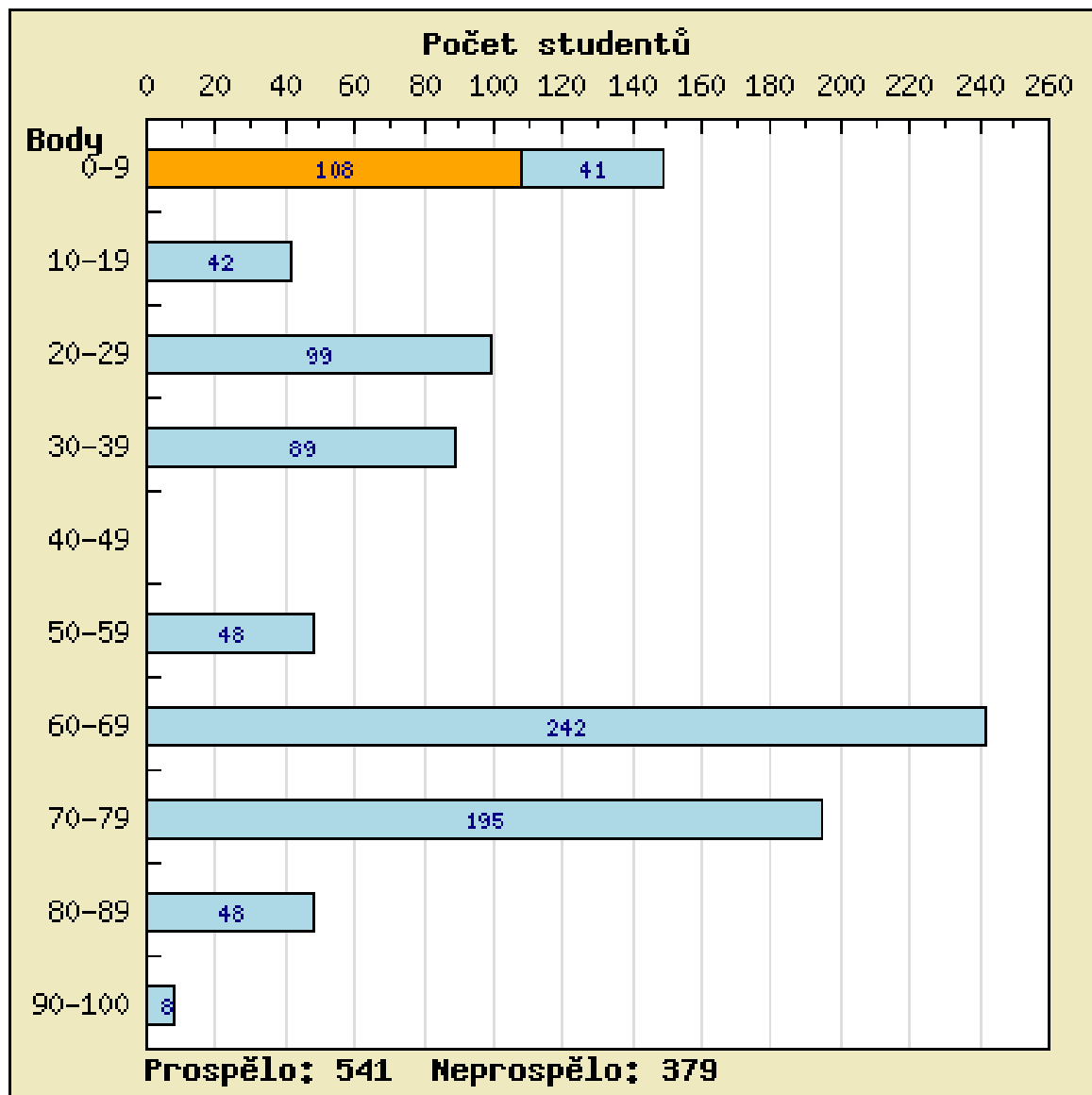


# Organizace předmětu – Hodnocení

- Cvičení: 12 bodů
- Domácí úloha: 12 bodů
- Projekt: 16 bodů
- Zápočet: 40 bodů (min. 18 bodů, tj. 45 %)
- Zkouška: 60 bodů (min. 30 bodů, tj. 50 %)
- Celkem: 100 bodů (min. 50 bodů, tj. 50 %)
- Pro přistoupení ke zkoušce je nutný zápočet.
- Vypíšeme 5 termínů zkoušky.
- Zkoušku lze 2× opakovat (jen při neúspěchu u předchozího termínu).

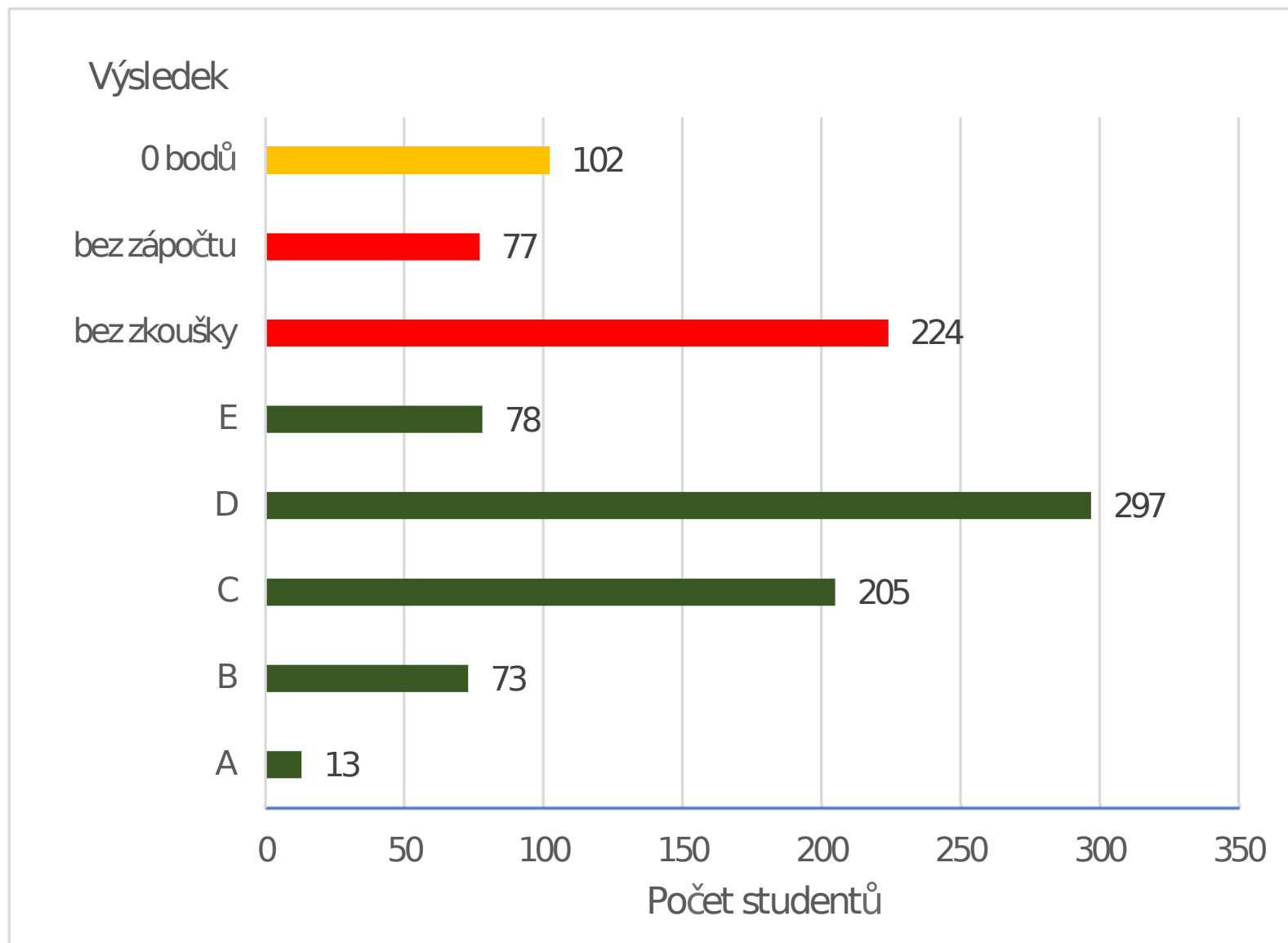
| Bodů     |   | Klasifikace | Číselně | Slovně       |
|----------|---|-------------|---------|--------------|
| 90 - 100 | ⇒ | A           | 1       | výborně      |
| 80 - 89  | ⇒ | B           | 1,5     | velmi dobře  |
| 70 - 79  | ⇒ | C           | 2       | dobře        |
| 60 - 69  | ⇒ | D           | 2,5     | uspokojivě   |
| 50 - 59  | ⇒ | E           | 3       | dostatečně   |
| 0 - 49   | ⇒ | F           | 4       | nevyhovující |

# Histogram hodnocení 2021/2022 (jen FIT)



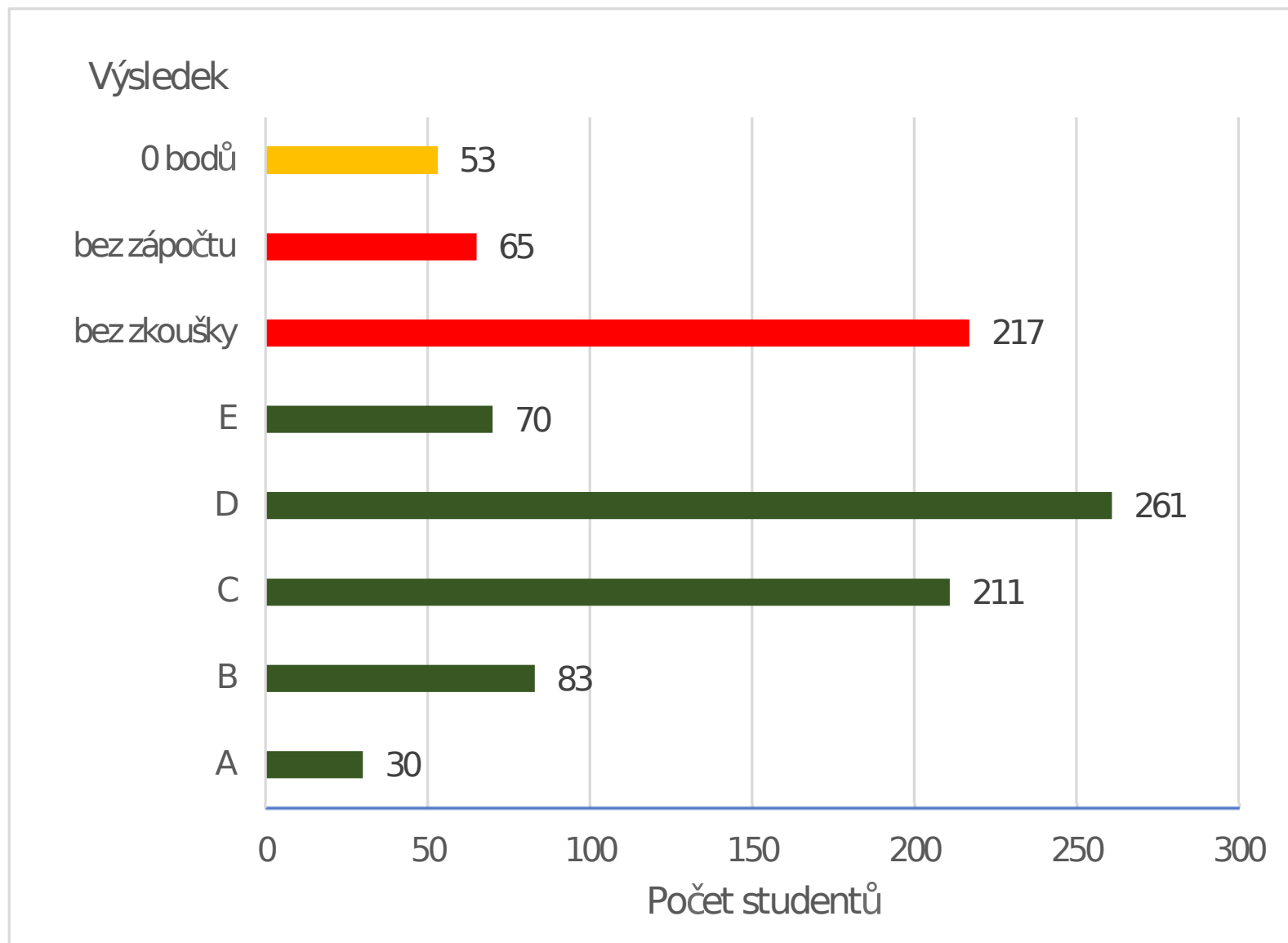
úspěšnost 66,6 %  
(bez neaktivních)

# Histogram hodnocení 2022/2023



úspěšnost 68,9 % (bez neaktivních)

# Histogram hodnocení 2023/2024



úspěšnost 69,9 % (bez neaktivních)

# Organizace předmětu – Komunikace

- **Informační systém VUT – STUDIS**

- obecné informace o předmětu
- rozvrhy
- termíny zkoušek
- celkové hodnocení

- **E-learning – systém Moodle**

- <https://moodle.vut.cz/course/view.php?id=281002>
- plán přednášek včetně prezentací
- diskuzní fóra
- zadání a hodnocení domácích úloh a projektu
- studijní materiály

# Organizace předmětu – Návaznosti

- **Předmět Databázové systémy (IDS)**

- ER diagramy pro návrh databáze
- implementace projektu (IS) podle projektu z IUS

- **Ostatní předměty**

- diagramy UML pro návrh
- řízení týmových projektů
- problémy s časem při dokončování projektů

- **Státní závěrečná zkouška – tematické okruhy**

- 33. Životní cyklus softwaru (charakteristika etap a základních modelů).
- 34. Jazyk UML.
- 35. Konceptuální modelování a návrh relační databáze.
- 40. Objektová orientace (základní koncepty, třídě a prototypově orientované jazyky, OO přístup k tvorbě SW).
- <https://www.fit.vut.cz/fit/info/rozhodnuti/2024/rd02-240110.pdf>

- **Praxe**

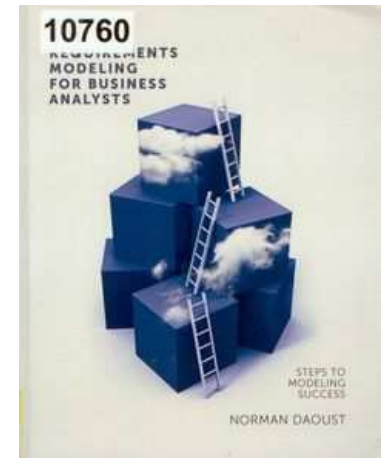
# Cíle předmětu

- získat základní přehled v oblasti tvorby rozsáhlých softwarových systémů,
- seznámit se s procesem tvorby softwaru a s etapami jeho životního cyklu,
- naučit se používat základní modely UML.

# Literatura

- N. Daoust. **UML Requirements Modeling For Business Analysts.**

Základní přehled UML diagramů  
a způsobu jejich použití.



- K. Wieggers, J. Beatty. **Software Requirements.**

Proces získávání, specifikace  
a validace požadavků.

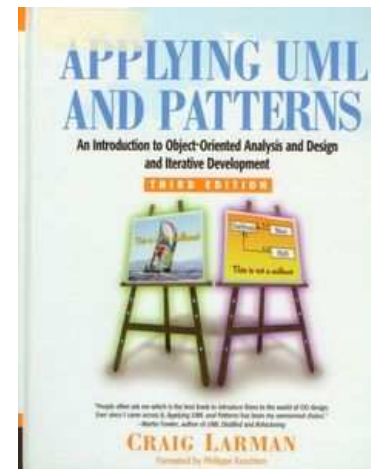




# Literatura

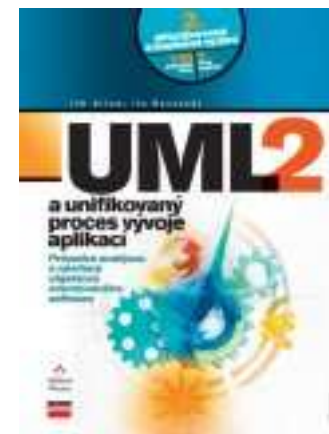
- C. Larman. **Applying UML and Patterns.**

Proces vývoje softwaru,  
základní modely, metodiky, UML diagramy,  
principy OO návrhu, návrhové vzory.



- J. Arlow. **UML 2 a unifikovaný proces vývoje.**

UML diagramy v procesu vývoje,  
zaměřeno na UP.



# Co je to softwarové inženýrství?

# Co je to softwarové inženýrství?

- systematický přístup k vývoji, nasazení a údržbě softwaru  
*The application of a systematic, disciplined, quantifiable approach to development, operation, and maintenance of software; that is, the application of engineering to software. (IEEE Standard Computer Dictionary, 1990)*
- inženýrská disciplína zabývající se praktickými problémy vývoje rozsáhlých softwarových systémů (Vondrák, 2002)

# Co je to softwarové inženýrství?

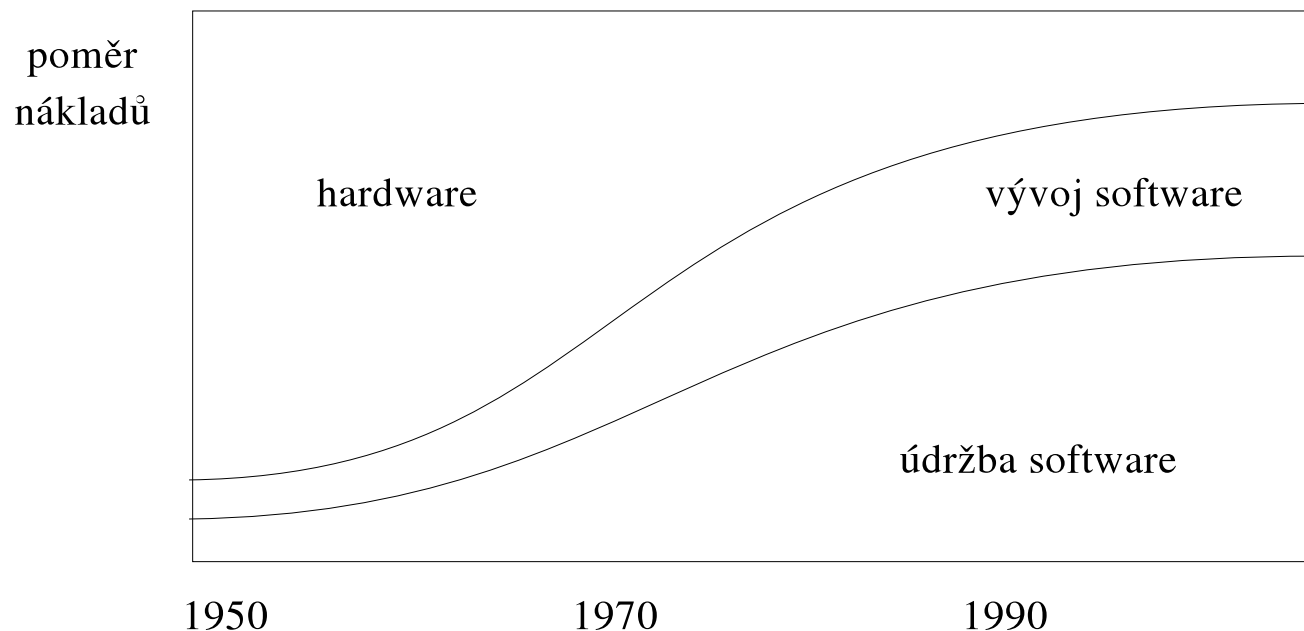
- systematický přístup k vývoji, nasazení a údržbě softwaru  
*The application of a systematic, disciplined, quantifiable approach to development, operation, and maintenance of software; that is, the application of engineering to software. (IEEE Standard Computer Dictionary, 1990)*
- inženýrská disciplína zabývající se praktickými problémy vývoje rozsáhlých softwarových systémů (Vondrák, 2002)

! softwarové inženýrství  $\neq$  programování

zkouma vylepsuje vyvoj a nasazeni a udrzbu systemu snazi se o co nejvetsi systematicnost aby zabranil zbytecnym nakladum a nedej boze katastrofam duvod vzniku discipliny jako takove je neustavajici rust nakladu na software

# Proč softwarové inženýrství?

- Proč vytváříme software?
  - zlepšení služeb – informační systémy, ...
  - snížení nákladů – řízení výroby, ...
  - nemožnost řešení bez použití počítačů – předpověď počasí, ...
- Je nutné zlepšovat vlastnosti SW, hlavně jeho spolehlivost, bezpečnost a použitelnost.
- Je potřeba zvyšovat produktivitu vývoje SW.



# Proč softwarové inženýrství?

## Katastrofy (málem) způsobené SW chybou

- 1996: Přetečení při konverzi 64 b. čísla v plovoucí řádové čárce na 16 b. celé číslo se znaménkem reprezentující vertikální rychlost vedlo 40 s po startu k autodestrukci rakety Ariane 5.
- 1985-1987: V důsledku odstranění hardwarové zábrany proti nadměrnému ozáření při vývoji lékařského přístroje Therac-25 a SW chyb bylo nadměrně ozářeno 6 pacientů (3 na následky zemřeli). Varující je zejména přístup výrobce, který při prvních případech nadměrného ozáření místo nápravy tvrdil, že k němu **nemůže** dojít.
- 1983: Sovětský systém pro včasné varování před nukleárním útokem nahlásil obsluze pět balistických střel mířících z USA na Moskvu. Operátor naštěstí použil hlavu (pokud by USA zaútočily na Sovětský svaz, použily by více než pět raket) a vyhodnotil to jako falešný poplach (odrazy slunce od mraků) místo odvety.
- 2018/2019: Boeing 737 MAX 8
- Další informace jsou např. na URL:

<http://www5.in.tum.de/~huckle/bugse.html>

# Počátek SW inženýrství

Počátek – šedesátá léta 20. století

- problémy při vývoji větších programů
- zavedení pojmů *softwarové inženýrství* a *softwarová krize* na konferencích v letech 1968-1969
- *SW krize* se projevovala (a stále projevuje)
  - neúnosným prodlužováním a prodražováním projektů
  - nízkou kvalitou výsledných produktů
  - problematickou údržbou a inovacemi
  - špatnou produktivitou práce programátorů
  - řada projektů končila neúspěchem
- první kroky k metodickému přístupu k programování – strukturované programování

vyvoj software ma strasnou spoehlivost a nejde  
v ne mit duveru ze dodrzi co slibil ze bude delat

# (Ne)úspěšnost SW projektů (Standish Group Report, USA, 1995)

| Překročení nákladů o | Projektů      |
|----------------------|---------------|
| <b>méně než 20 %</b> | <b>15,5 %</b> |
| 21 - 50 %            | 31,5 %        |
| 51 - 100 %           | 29,6 %        |
| 101 - 200 %          | 10,2 %        |
| 201 - 400 %          | 8,8 %         |
| více než 400 %       | 4,4 %         |

| Překročení času o    | Projektů      |
|----------------------|---------------|
| <b>méně než 20 %</b> | <b>13,9 %</b> |
| 21 - 50 %            | 18,3 %        |
| 51 - 100 %           | 20,0 %        |
| 101 - 200 %          | 35,5 %        |
| 201 - 400 %          | 11,2 %        |
| více než 400 %       | 1,1 %         |



# (Ne)úspěšnost SW projektů (Standish Group Report, USA, 1995)

| Výsledná funkčnost | Projektů     |
|--------------------|--------------|
| méně než 25 %      | 4,6 %        |
| 25 - 49 %          | 27,2 %       |
| 50 - 74 %          | 21,8 %       |
| 75 - 99 %          | 39,1 %       |
| <b>100 %</b>       | <b>7,3 %</b> |

Průměrný SW projekt tedy v porovnání s původním plánem:

- stál o **89 %** více,
- trval **2,22 krát** déle a
- poskytuje pouze **61 %** funkčnosti.

**Průměrný projekt byl tedy téměř 7 krát horší, než se původně plánovalo!**

# Problémy při vývoji softwaru

spatná prezentace  
požadavky na  
neustále se měnící  
absurdní komplexnost  
a tím pádem  
neflexibilita

Podstatné, vnitřní, nevyhnutelné **problémy**:

- **Složitost** – žádné dvě části nejsou stejné; složitost je zdrojem dalších problémů jako např. komunikace v týmech; je náročné pochopit všechny možné stavy systému; problémy s úpravami a rozšířeními, ...
- **Přizpůsobivost** – když se něco změní, měl by se přizpůsobit software a ne naopak.
- **Nestálost** – mění se okolí a mění se i software (nejde o nahrazení novým); přibývají požadavky na úspěšně používaný software; software přežívá hardwarové prostředky.
- **Neviditelnost** – neexistuje přijatelný způsob reprezentace softwarového výrobku, který by pokryl všechny aspekty; dokonce ani nejsme schopni určit, co v dané reprezentaci chybí.

Syndrom *90% hotovo*: Při posuzování hotové části se nevychází z hotového, ale z odpracovaného (např. podle plánu).

# Problémy při vývoji softwaru

Problémy, které se nemusí projevit vždy:

- **práce v týmu**

- problémy s organizací práce na velkých softwarových projektech
- problémy s plánováním procesu tvorby softwaru
- Komunikační problémy jsou jedním z hlavních zdrojů chyb v programech.
- extrémní odchylky v produktivitě mezi jednotlivými programátory, až 1:20

- **nízká znovupoužitelnost při tvorbě softwaru**

- V procesu tvorby softwaru je málo standardů a většinou se software tvoří od začátku. S každým programem se vymýšlí už vymyšlené.
- Málo produktů se sestavuje z už existujících součástí.

- **problém míry**

- Metody použitelné na řešení malých problémů se nedají přizpůsobit na řešení velkých (složitých) problémů.

# Problémy při vývoji softwaru

Problémy, které se nemusí projevit vždy:

- **tvorba dokumentace**

- Tvorba dokumentace je podobná tvorbě vlastního programu.
- enormní rozsah dokumentace co do kvantity i rozmanitosti  
Např. ve velkých vojenských softwarových projektech připadalo 400 anglických slov na každý příkaz v programovacím jazyce Ada.
- problémy s udržováním aktuálnosti dokumentace vzhledem ke změnám softwaru
- problémy s konzistencí a úplností dokumentace

- **náchylnost softwaru k chybám** zakaznik nejlepsi tester

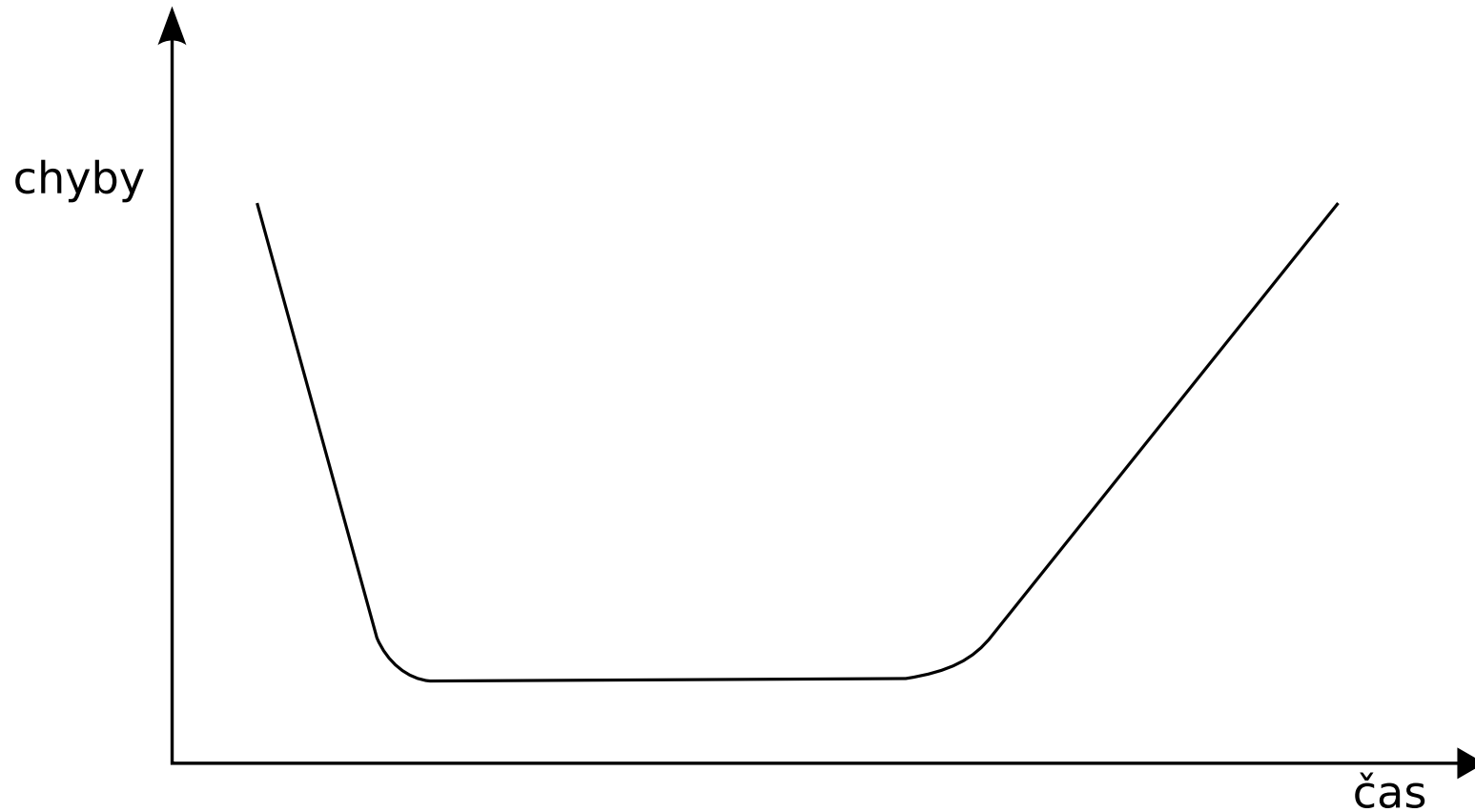
- Hodně chyb se projeví až při provozu (a ne při vývoji).
- Odstraňování chyb vede k návratu v etapách vývoje softwaru.

- **způsob stárnutí softwaru**

- Software se fyzicky neopotřebuje. ALE: Přidávání nových funkcí ve spojení s častými opravami chyb vede k postupné degradaci struktury a k snižování spolehlivosti softwarových systémů.

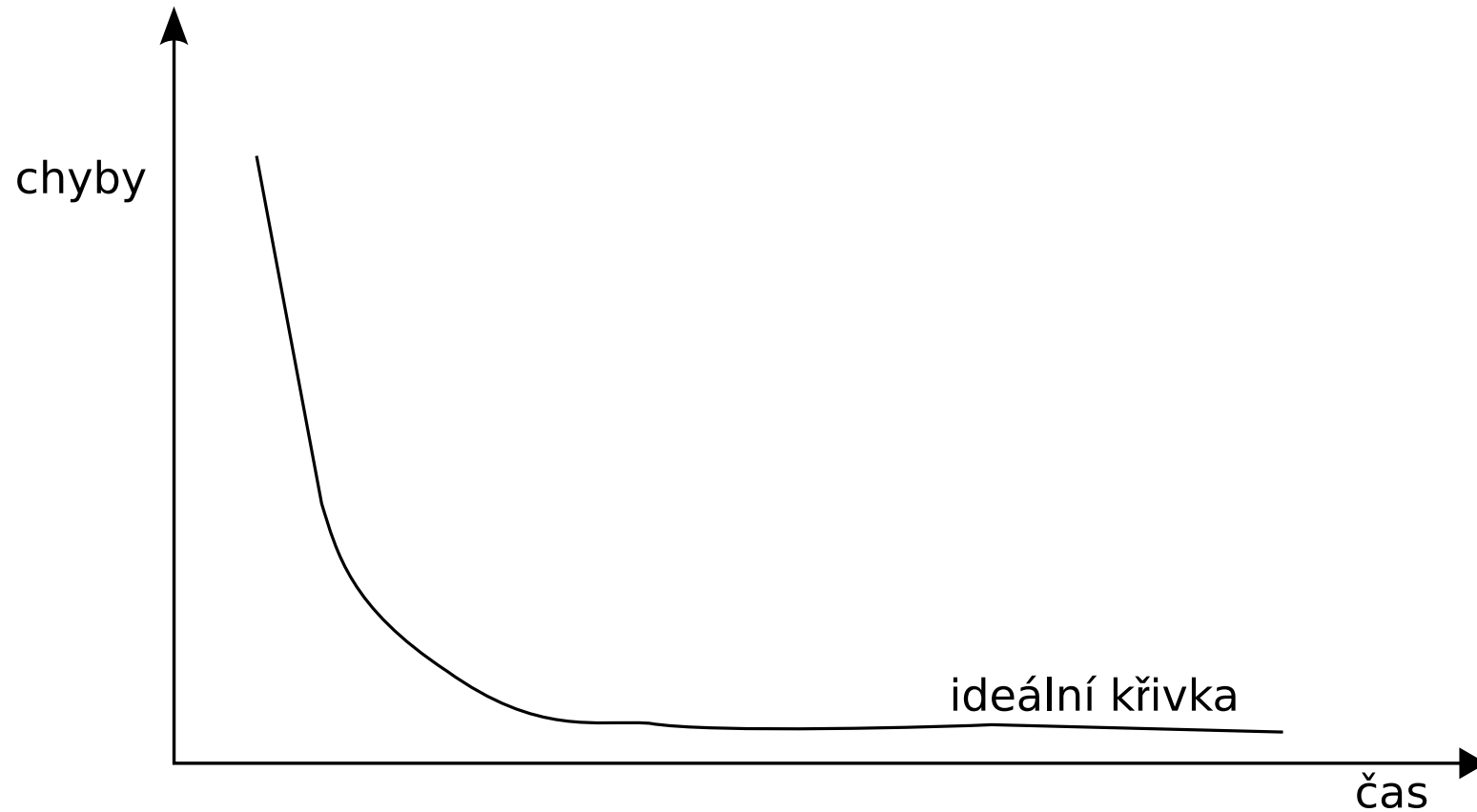
# Stárnutí hardwaru

## Typická chybová křivka hardwaru



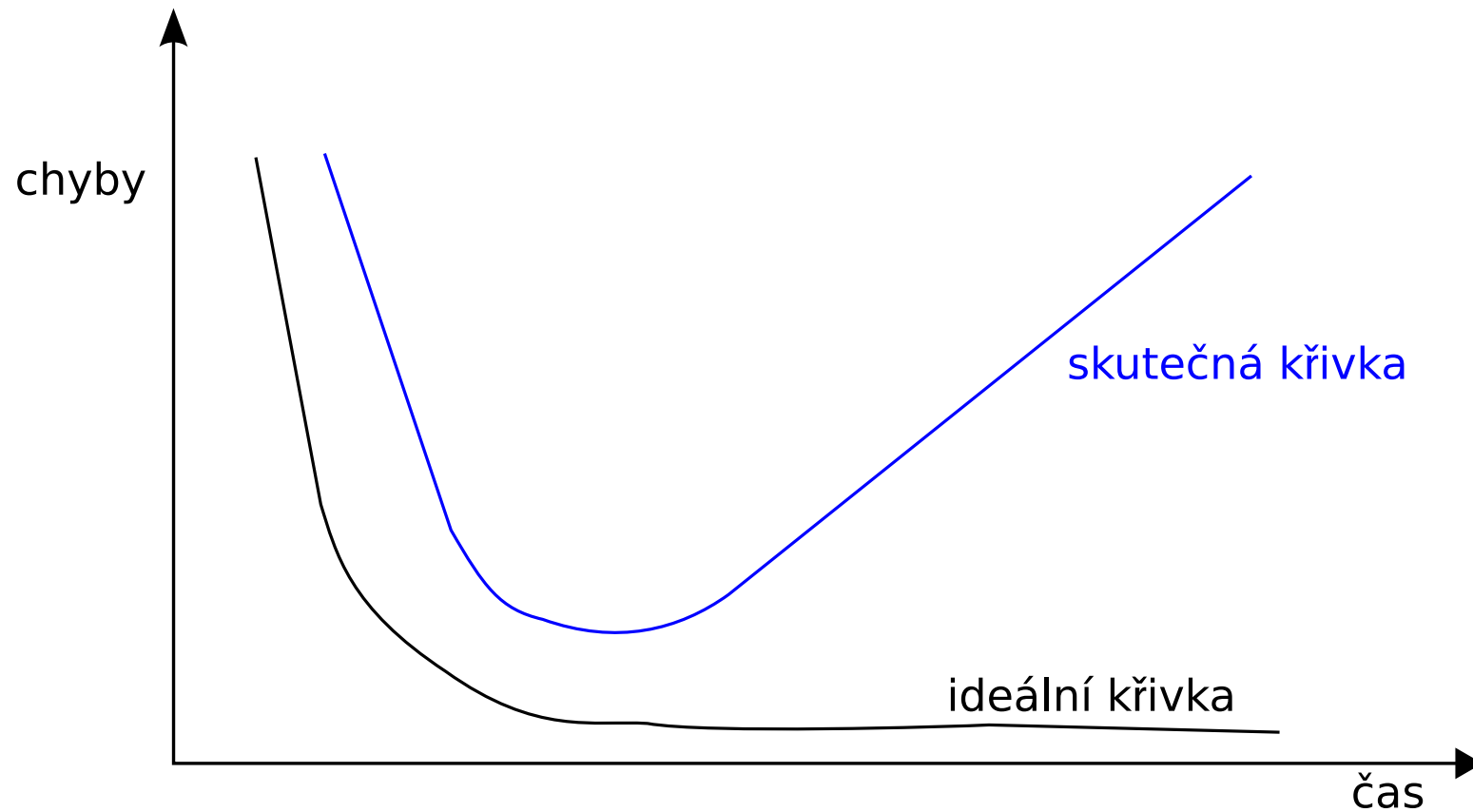
# Stárnutí softwaru

## Typická chybová křivka softwaru



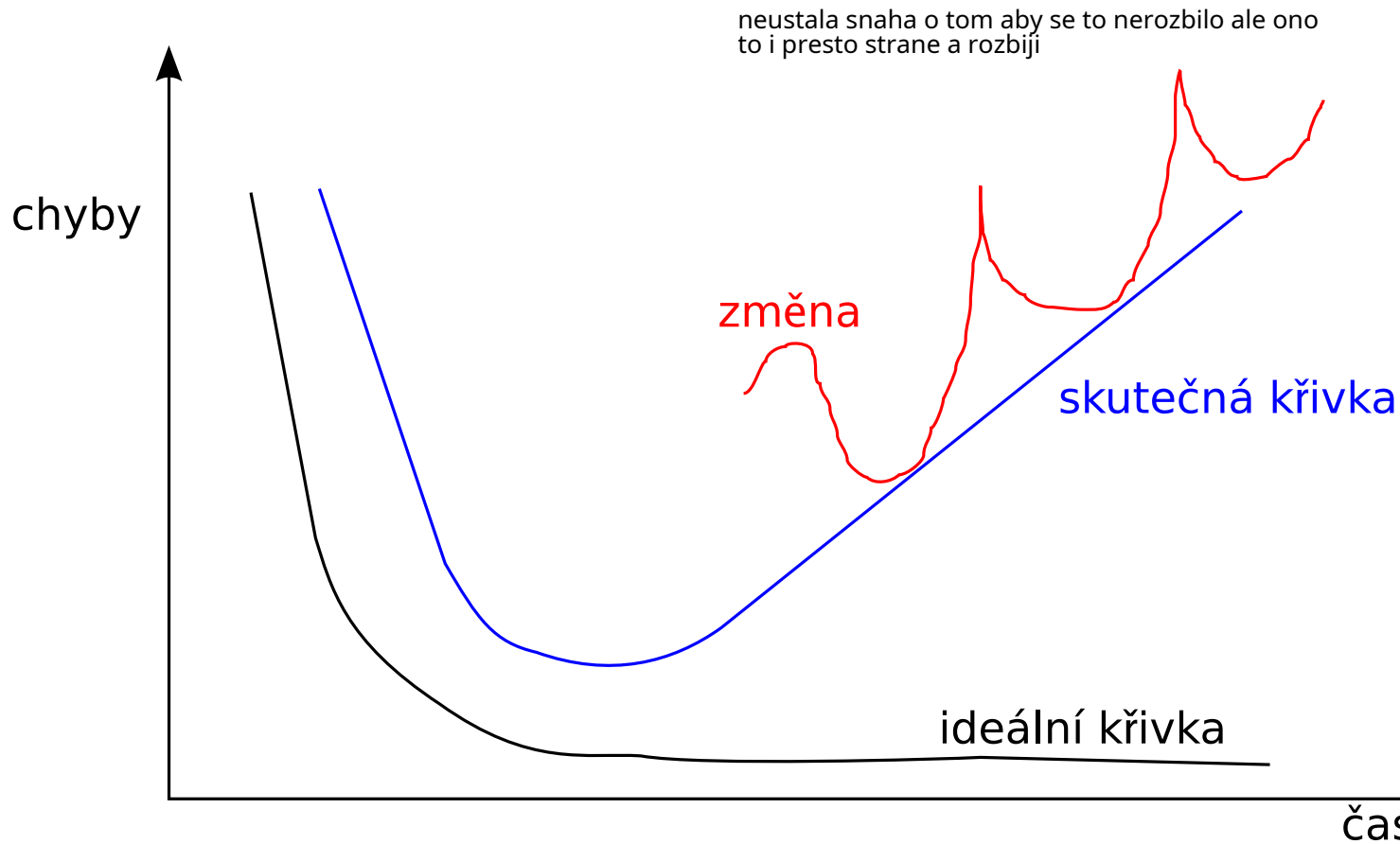
# Stárnutí softwaru

## Typická chybová křivka softwaru



# Stárnutí softwaru

## Typická chybová křivka softwaru





# Problémy při vývoji softwaru

Problémy, které se nemusí projevit vždy:

- **specifikace požadavků**
  - problematická komunikace s uživatelem
  - nejasná a neúplná formulace požadavků spojená s neucelenou představou uživatele o výsledném softwarovém systému
  - nejednoznačnost spojená s častou specifikací požadavků v přirozeném jazyce
  - ...

**Tvorba softwaru je tvůrčí proces, software nelze vyrábět.**

# Problémy při vývoji softwaru

## Příklad důsledku nepřesnosti či nepochopení specifikace.

*Personální oddělení (PO):* " Máme problém se systémem. Zaměstnankyně změnila jméno a systém změnu neakceptuje."

*IT oddělení (IT):* " Provdala se?"

*PO:* " Ne, pouze změnila jméno. Systém zřejmě vyžaduje změnu stavu osoby."

*IT:* " Ano, nikdy jsme neuvažovali, že by si někdo změnil jméno jen tak."

*PO:* " Předpokládali jsme, že víte, že lidé mohou kdykoliv legálně změnit jméno. Potřebujeme změnu jména zavést do systému, abychom mohli zadat výplatu. Kdy odstraníte chybu?"

*IT:* " To není chyba! Nevěděli jsme, že potřebujete tuto vlastnost. Můžeme tuto novou vlastnost zavést do konce měsíce. Příště nám své požadavky řekněte dříve."

*K. Wiegers, J. Beatty: Software Requirements. Microsoft Press, 2013.*

# Příčiny zastavení softwarových projektů

... podle analýzy víc jak 350 firem a 8000 aplikací:

- neúplnost nebo nejasnost požadavků (13,1 %)
- nedostatek zájmu a podpory ze strany uživatele (12,4 %)
- nedostatek zdrojů, tj. podhodnocený rozpočet a krátké termíny (10,6 %)
- nerealistické očekávání (9,9 %)
- malá podpora od vedení dodavatele nebo odběratele (9,3 %)
- změna požadavků a specifikace (8,7 %)
- nedostatečné plánování (8,1 %)
- vyvíjený systém už není potřeba (7,5 %)
- ...

# Pár postřehů Freda Brookse

- Přidáním dalších pracovníků do zpožděného projektu se tento projekt ještě více zpozdí.
- Napsání překladače Algolu zabere 6 měsíců nezávisle na tom, kolik ho vytváří programátorů.
- Efekt (syndrom) druhého systému – při návrhu druhé verze systému hrozí rizika:
  - příliš složitý a neefektivní systém

*Systém není dokonalý, když k němu nelze nic přidat, ale tehdy, když z něho nelze nic odstranit.*

- nepoužití nových technologií

vice lidi nerovna se rychlejsi vyvoj systemu

# Rozvoj SW inženýrství

- Výzkum programovacích praktit
  - uvědomění si lidského faktoru, práce v týmu
  - podpora řízení tvorby SW
  - modulární programování je potřeba umet pracovat v tymu nenejboze umet komunikovat
  - návrhové vzory
- Výzkum metodik
  - vnímání životního cyklu vývoje SW
  - strukturované metody, datově a procesně orientované metody, objektově orientované metody, agilní metodiky, ...
  - výzkum modelovacích jazyků (dnes UML)
- Zabezpečení kvality
  - systematické testování, formální ověřování
- Metody návrhu založené na modelech
  - transformace modelů do programu

# Metodiky vývoje softwaru

## Metodiky

- disciplinovaný proces nad vývojem softwaru s cílem zajistit tento vývoj více predikovatelný a efektivnější
- věnují se různým aspektům, které ovlivňují vývoj softwarového produktu, včetně samotného procesu tvorby softwaru
- zahrnují proces vývoje, nástroje, způsoby využití, plánování, ...

## Pozor na terminologii!

- *Metoda* – postup pro dosažení určitého cíle
- *Metodika* – souhrn doporučených praktik a postupů
- *Metodologie* – nauka o metodách, jejich tvorbě a použití

**Ale!**

Metodika vývoje softwaru = Software Development Methodology

# Softwarový produkt

## Program

- funkční část produktu
- softwarový produkt není jenom kód jako samotný ale dokumentace, postupy, pravidla, dokumentace a další spoje věci

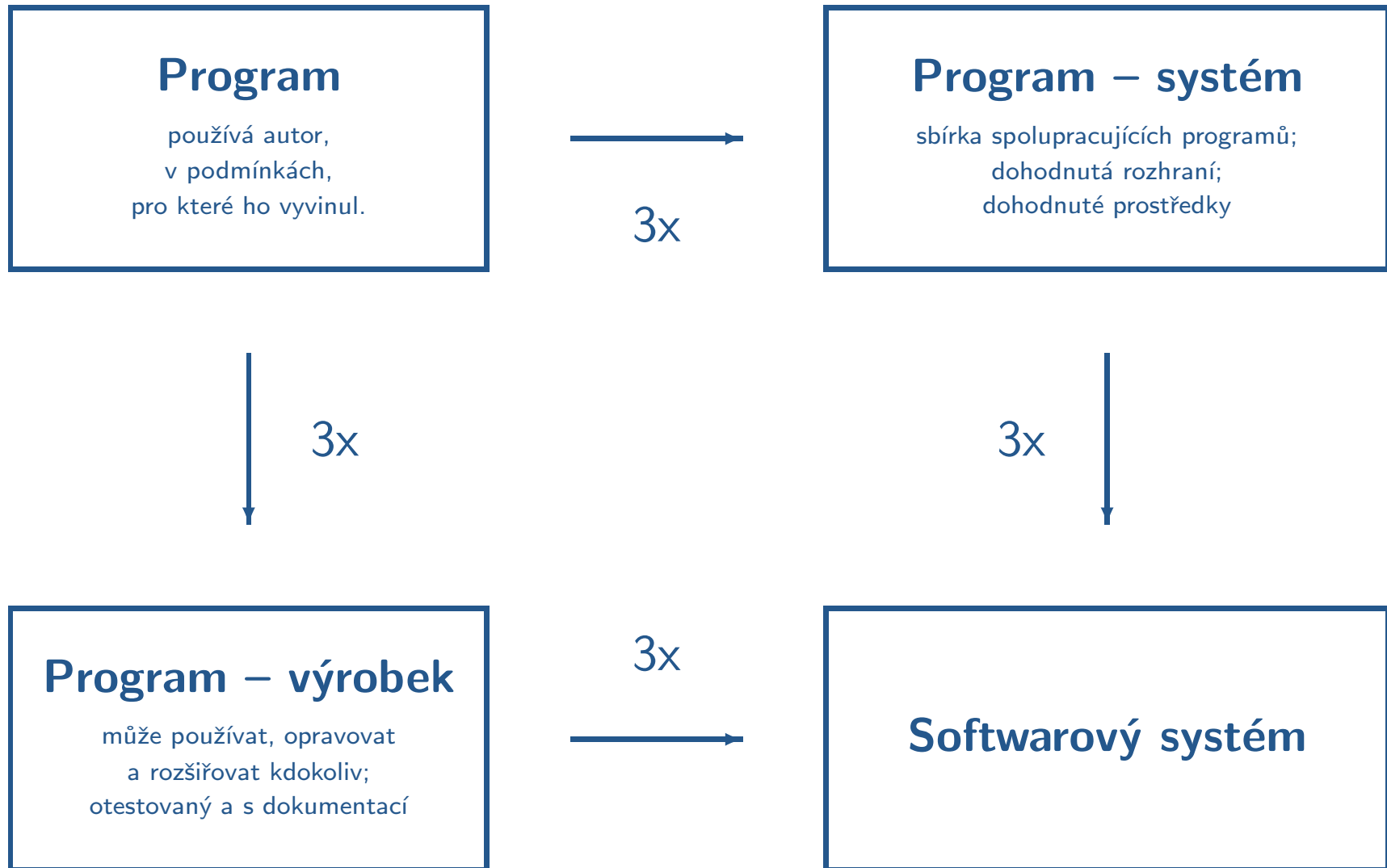
## Softwarový produkt

- sbírka počítačových programů, procedur, pravidel a s nimi spojená dokumentace
- zahrnuje např.: požadavky, specifikace, popisy návrhu, zdrojové texty, testovací data, příručky, ...

## Akteři ve vývoji softwarového produktu (softwaru)

- **Zákazník** – sponzoruje vývoj SW, specifikuje požadavky na SW
- **Dodavatel** – vyvíjí systém, má závazky vůči zákazníkovi, komunikuje s uživatelem (testování, ...)
- **Uživatel** – testuje a používá systém, upřesňuje požadavky na SW

# Vztah mezi programem a softwarem





# Typy softwarových produktů

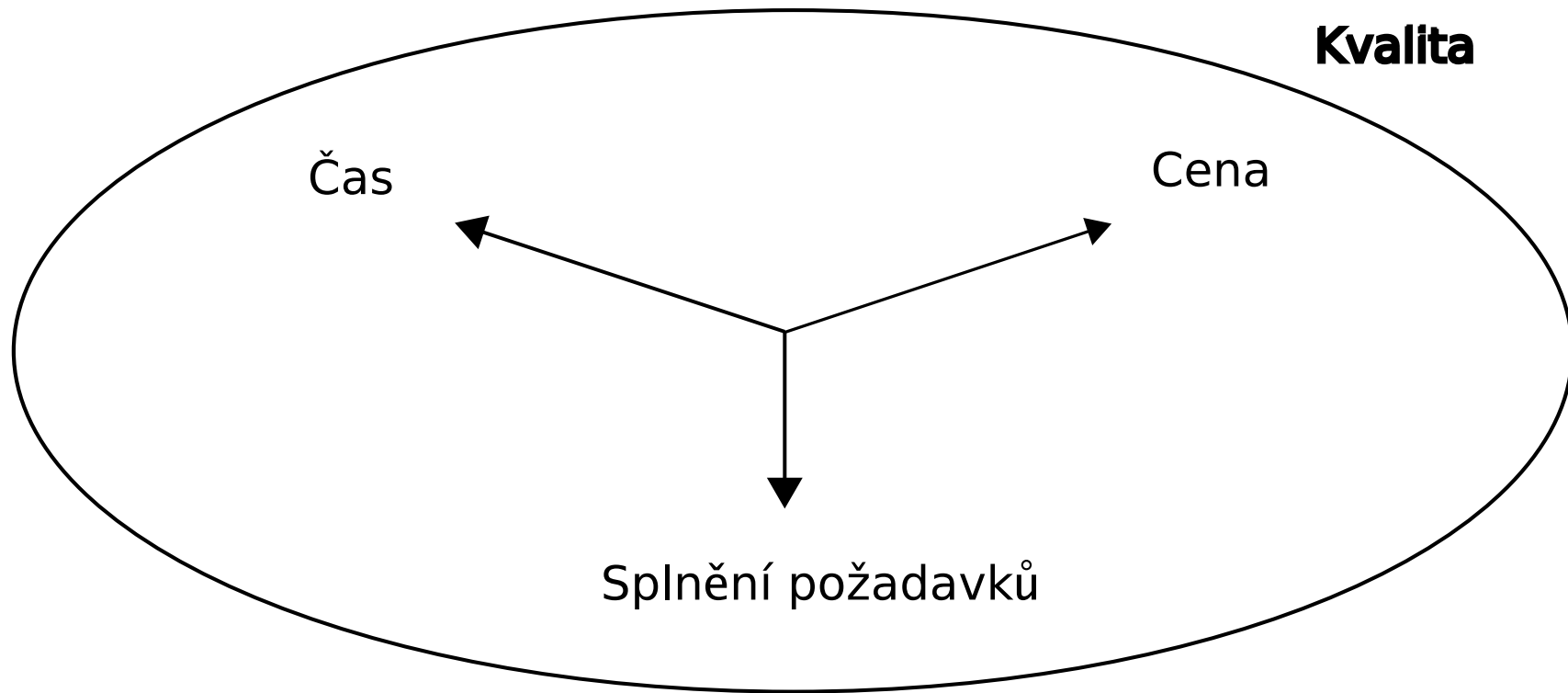
## Generické

- Software se prodává libovolnému zájemci (krabicový software).
- Musí být velice důkladně otestován, protože opravy chyb jsou vzhledem k velkému rozšíření drahé.

## Zákaznické (na objednávku)

- Software se vytváří na základě požadavků pro konkrétního zákazníka.
- Většinou pro specializované aplikace, pro které vhodný generický software neexistuje.
- Cena zákaznického softwaru je výrazně vyšší.
- Dvě možnosti jeho tvorby:
  - zadáním zakázky SW firmě
  - v rámci vlastní firmy

# Kvalita SW produktů



# Proces vývoje softwaru

Proces, ve kterém

- se potřeby uživatele transformují na požadavky na SW,
- požadavky na SW se transformují na návrh,
- návrh se implementuje,
- implementace se testuje
- a nakonec předá uživateli.

SW proces definuje

- kdo
- dělá co
- a kdy
- $\Rightarrow$  jak dosáhnout požadovaného cíle

# Životní cyklus softwaru

## Životní cyklus

- rozděluje proces vývoje softwaru na za sebou jdoucí období
- pro každé období stanovuje cíl
- období = **etapa životního cyklu softwaru**

## Činnosti spojené s vývojem softwaru

- analýza a specifikace požadavků (8 %), nejdůležitější část
- architektonický a podrobný návrh (7 %),
- implementace (12 %),
- integrace a testování (6 %),
- provoz a údržba (67 %).

Úsilí věnované pečlivé analýze a návrhu se vrátí úsporou nákladů později.

# Etapy životního cyklu softwaru

## Analýza a specifikace požadavků

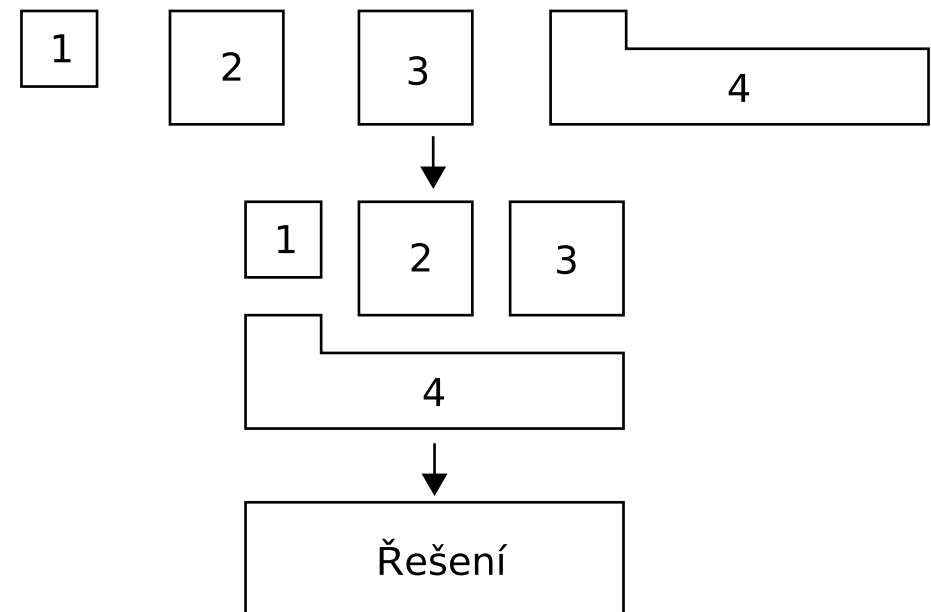
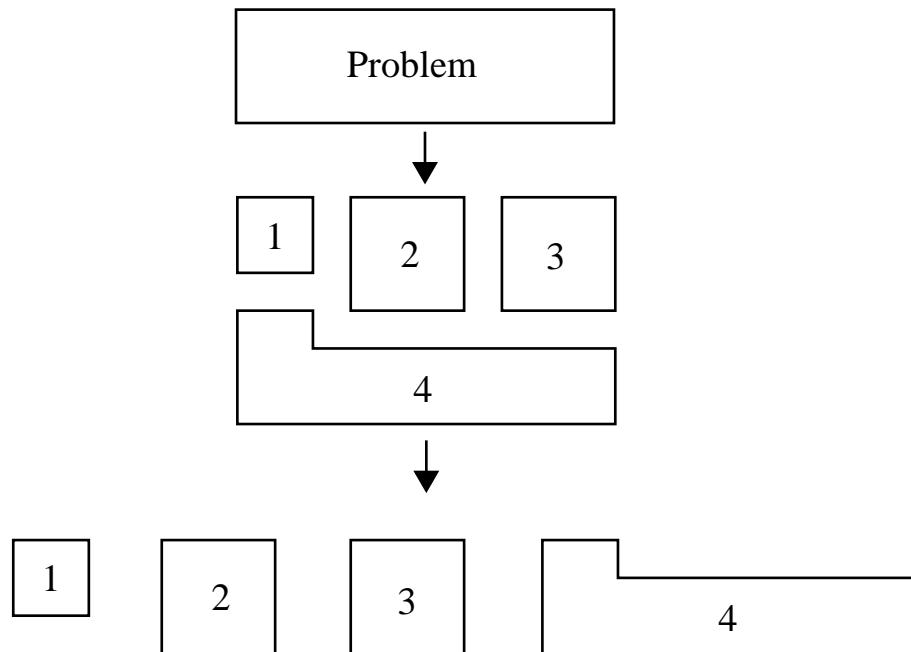
- získávání, analýza, definování a specifikace požadavků  $\Rightarrow$  transformace neformálních požadavků uživatele do strukturovaného popisu požadavků,
- cílem je identifikovat požadavky uživatele, ne návrh, jak je realizovat,
- provedení studie vhodnosti, identifikace a analýza rizik,
- plánování akceptačního testování.

nejnarocnejsi pasaz jelikoz casto zakaznik nevi co chce a zaroven vyvoj neni schopen mu naslochat a nebo to pochopi jinak nez zakaznik myslel

uvedomeni jestli jsme toho schopni vlastne

# Dekompozice složitých problémů

- rozdělení (dekompozice) složitějšího problému na jednodušší (lehčí zvládnutí problému)
- rozhraní podsystémů



# Dekompozice složitých problémů

## Přináší

- lépe zvládnutelné podsystémy
- soustředění pozornosti na jeden podsystém
- prezentovatelnost dílčího problému bez rušivých vlivů
- podsystémy se mohou vyvíjet nezávisle
- **skutečně velké systémy se bez dekompozice nedají zvládnout**

když systém rozdělíme na jednodušší části tak se na každou část může vyladit jiný tým a nebudou dělat všichni všechno najednou

## Zvýšená pozornost

- koordinace tvorby rozhraní
- integrace a testování podsystémů

zrovna není lehké uhlídat aby se podsystémy nerozcházeli a byli spolu schopni spolupracovat

# Etapy životního cyklu softwaru

## Architektonický návrh

- ujasnění koncepce systému,
- dekompozice systému,
- definování vztahů mezi částmi systému,
- specifikace funkcionality a ohraničení podsystémů,
- plánování testování systému,
- plánování nasazení systému do provozu, dohoda o postupu nasazování podsystémů, dohoda o plánu zaškolování uživatelů.



# Etapy životního cyklu softwaru

## Podrobný návrh

- podrobná specifikace softwarových součástí,
- specifikace algoritmů realizujících požadované funkce,
- specifikace rozhraní pro jednotlivé součásti,
- specifikace logické a fyzické struktury údajů, které zpracovává příslušná součást,
- specifikace způsobu ošetřování chybových a neočekávaných stavů,
- plán prací při implementaci součástí,
- plán testování součástí, návrh testovacích dat,
- specifikace požadavků na lidské zdroje (odhad trvání a nákladů projektu).

# Etapy životního cyklu softwaru

## Implementace a testování součástí

- programová realizace softwarových součástí,
- vypracování dokumentace k součástem,
- testování implementovaných součástí,
- začátek školení budoucích uživatelů.

## Integrace a testování systému

- spojení součástí do podsystémů,
- testování podsystémů,
- integrace podsystémů do celého systému,
- testování podsystémů a celého systému  
oprava nalezených chyb, návraty k etapě implementace.

# Etapy životního cyklu softwaru

## Akceptační testování a instalace

- testování systému uživatelem,
- operace přebírání SW produktu,
- školení používání systému, nasazení systému.

## Provoz a údržba

- zabezpečení provozu softwaru,
- řešení problémů s nasazením softwaru,
- řešení problémů s používáním softwaru,
- opravy, rozšiřování, přizpůsobování softwaru podle požadavků okolí.

# Model životního cyklu softwaru

## Model životního cyklu

- definuje etapy vývoje softwaru a jejich časovou následnost,
- pro každou etapu definuje nutné činnosti,
- pro každou etapu definuje její vstupy a výstupy.

rika pouze co se v te dane etape ma odehrat a jake z toho maji byt vysledky ale nerika jak dlouha ma kazda etap trvat

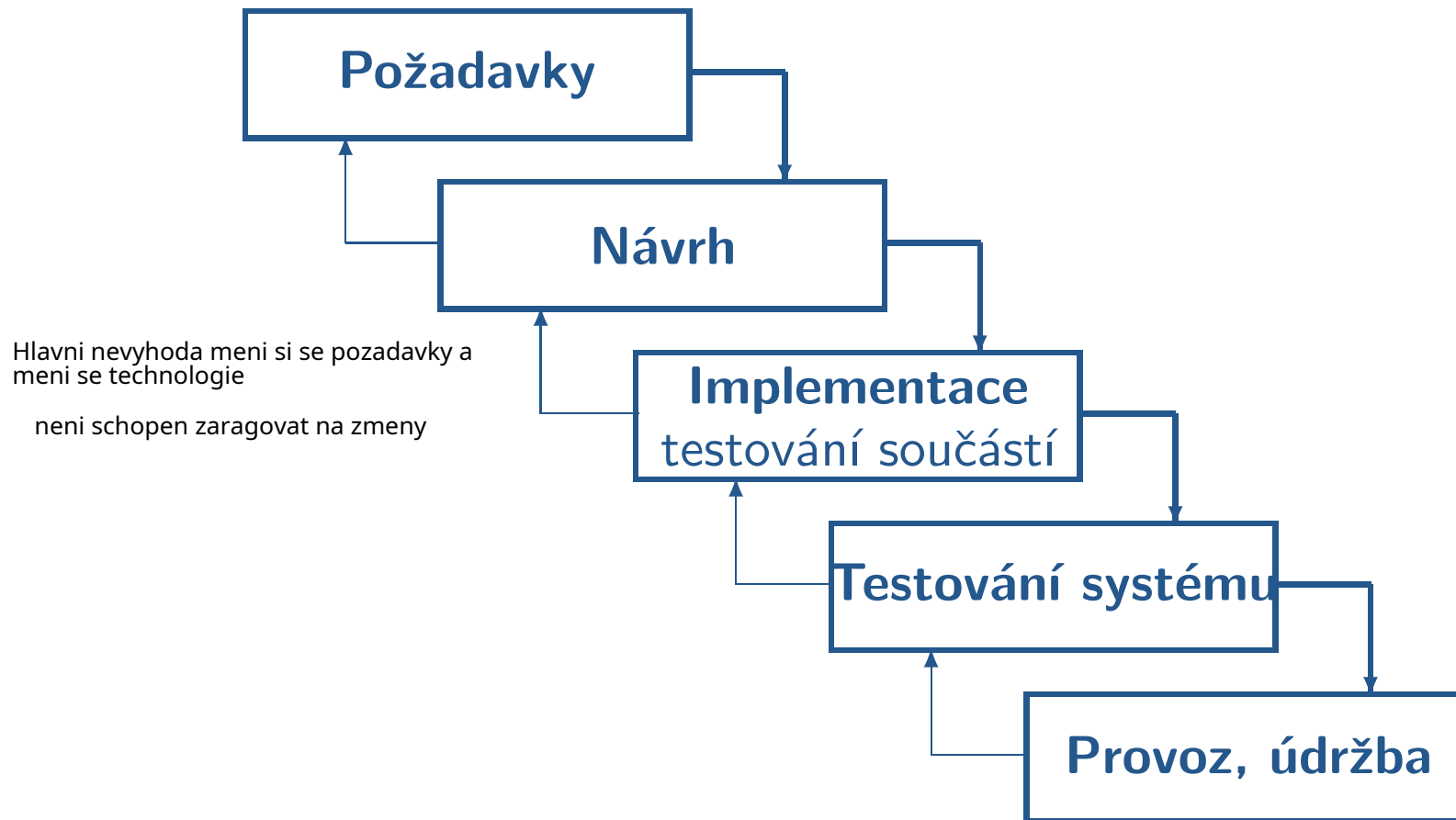
## Další vlastnosti

- nedefinuje délku trvání kroků a jejich rozsah,
- každá etapa vytváří reálné výstupy,
- ~~správnost každé etapy lze vyhodnotit.~~

Rozdíly v modelech jsou zejména v definování etap a jejich posloupnosti.

# Vodopádový model životního cyklu softwaru

- životní cyklus jde postupně od první etapy až do poslední
- následující etapa začne až po ukončení předcházející
- možnost návratu k předchozí etapě



# Vodopádový model

## Vlastnosti

- lineární (sekvenční) model, intenzivně používán v 70. letech
- cílem bylo zavést do vývoje řád umožňující řešit náročnější problémy
- dekompozice, kontrola výstupů etap  $\Rightarrow$  snížení počtu chyb
- uživatel se účastní pouze při definování požadavků a zavádění

## Výhody

- jednoduché na řízení
- při stálých požadavcích: nejlepší struktura výsledného produktu

## Nevýhody

- zákazník není schopen předem stanovit (přesně!) všechny požadavky
- při změnách požadavků dlouhá doba realizace
- zákazník vidí spustitelnou verzi až v závěrečných fázích projektu  $\Rightarrow$  odhalení nedostatků ve specifikaci požadavků příliš pozdě (validace)

# Hlavní cíle SW inženýrství

- **Management projektu**
  - řízení životního cyklu projektu
  - dosažení požadovaného výsledku v požadovaném čase
  - $\Rightarrow$  efektivní práce s časem a tedy i s náklady
- **Techniky**
  - analýzy
  - návrhu
  - programování
  - testování
  - ...
- **Vlastnosti SW inženýra**
  - základní báze znalostí
  - schopnost aplikovat znalosti
  - schopnost vyhledávat nové informace a osvojit si nové znalosti
  - ...

# Studijní koutek

- Měl by vám pomoci s orientací při studiu.
- Je pro něj vyhrazeno posledních 10 minut přednášky.
- Zde uvedené informace se nezkoušejí.
- Můžete posílat náměty na to, co vás zajímá.



# Vysoké učení technické v Brně

- **Historie**

- 1849 – německo-české technické učiliště
- **1899** – Česká vysoká škola technická v Brně
- 1956 – Vysoké učení technické v Brně

- **Vedení**

- Nejvyšším představitelem vysoké školy je rektor.
- 53. rektorem je **doc. Ing. Ladislav Janíček, Ph.D., MBA, LL.M.**

- **Fakulty**

- Fakulta architektury – FA
- Fakulta elektrotechniky a komunikačních technologií – FEKT
- Fakulta chemická – FCH
- **Fakulta informačních technologií – FIT**
- Fakulta podnikatelská – FP
- Fakulta stavební – FAST
- Fakulta strojního inženýrství – FSI
- Fakulta výtvarných umění – FAVU

# Fakulta informačních technologií

- **Historie**

- 1964 – Katedra samočinných počítačů na FE
- 1990 – Katedra informatiky a výpočetní techniky na FE
- 1992 – Ústav informatiky a výpočetní techniky na FE
- 1993 – reorganizace FE  $\Rightarrow$  FEI
- 2002 – Fakulta informačních technologií (FIT)

- **Ústavy**

- Ústav informačních systémů
- Ústav inteligentních systémů
- Ústav počítačové grafiky a multimédií
- Ústav počítačových systémů

# Fakulta informačních technologií

- **Vedení**

- Nejvyšším představitelem fakulty je děkan.
  1. děkanem byl (2002–2008) **prof. Ing. Tomáš Hruška, CSc.**
  2. děkanem byl (2008–2016) **doc. Ing. Jaroslav Zendulka, CSc.**
  3. děkanem byl (2016–2024) **prof. Dr. Ing. Pavel Zemčík, dr. h. c.**
  4. děkanem je (od 2024) **doc. Dr. Ing. Petr Hanáček.**
- Proděkanem pro vzdělávací činnost
  - v bakalářském studiu je **doc. Ing. Radek Burget, Ph.D.**
  - v magisterském studiu je **doc. Ing. Richard Růžička, Ph.D., MBA.**
- Studijní poradce je **Ing. Petr Veigend, Ph.D.**