Softwarové inženýrství

1. přednáška

Radek Janoštík

Univerzita Palackého v Olomouci

25. 9. 2025

Outline

Úvod

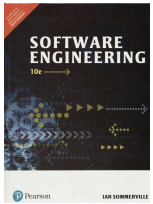
- Základy vývoje SW s inženýrského/manažerského pohledu
- Živelný vývoj není správnou cestou $\Rightarrow$ různé (osvědčené) metody vedení projektů
- 3 hodiny týdně přednáška + 2 hodiny cvičení
- Cvičící: Mgr. Jiří Zacpal Ph. D.

Konzultace, kontakt

- Email: radek.janostik@upol.cz
- Pracovna: 5.073
- Telefon: 585 634 711
- Web: `https://apollo.inf.upol.cz/~janostik/`
- Konzultace: úterý 11:45 – 13:00

Doporučená literatura (1 / 3)

- Ian Sommerville. Software Engineering (10th Edition). Pearson, 2015. ISBN: 978-93-325-8269-9.



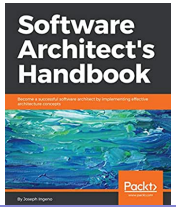
  - Případně její starší edice

- Steve McConnel. Dokonalý kód. Computer Press, 2005. ISBN: 80-251-0849-X

Doporučená literatura (2 / 3)

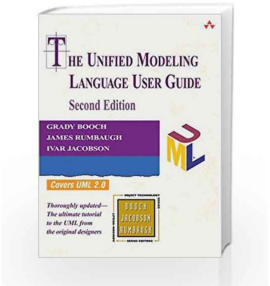
- Karl Wiegers. Požadavky na software. Computer Press, 2008. ISBN: 978-80-251-1877-1

- Joseph Ingeno. Software Architect's Handbook. Packt, 2018. ISBN: 978-1-78862-406-0



Doporučená literatura (3 / 3)

- Grady Booch, James Rumbaugh, Ivar Jacobson. The Unified Modeling Language User Guide (2th edition). Addison Wesley, 2005. ISBN: 321267974



Jízdní řád (1/2)

- 25. 9. 2025 – Úvodní hodina, system engineering
- 2. 10. 2025 – Softwarový proces
- 9. 10. 2025 – Jazyk UML
- 16 .10. 2025 – Softwarové požadavky, specifikace, funkční, nefunkční, doménové
- 23. 10. 2025 – Design/architektura aplikací
- 30. 10. 2025 – Návrhové vzory
- 6. 11. 2025 – Testování

Jízdní řád (2/2)

- 13. 11. 2025 – Kvalitní kód
- 20. 11. 2025 – Bezpečnost, sw chyby, následky, příklady
- 27 .11. 2025 – Správa verzí
- 4. 11. 2025 – CI/CD
- 11. 12. 2025 – Rezerva
- 18. 12. 2025 – Zápočtový týden
- Změny v plánu vyhrazeny

Zkouška

- Zkouška bude ústní formou
- Dvě otázky
  - 1. „do hloubky“ (komplexnější téma)
  - 2. „přehledová“ (vedlejší téma, přehledově, základní principy)
- Otázky z probraných okruhů
  `https://apollo.inf.upol.cz/~janostik/soft/` (mohou být ještě upraveny)
- $\approx$ 20 minut příprava $\Rightarrow$ ústní zkoušení, doptávání, diskuze
- Předtermín v zápočtovém týdnu

Dotazy?

- Aktivní účast
- Přestávka?

Anketa

- Kdo už někdy vyvíjel nějakou větší aplikaci/projekt?
  - Co je „větší“ aplikace?
  - Co vlastně určuje velikost projektu?
- Jak se vám dělal plán vývoje?
  - Co časový odhad prací?
- Kdo již někdy něco vyvíjel v týmu?
  - Jaká úskalí to přináší?
- Prodal již někdo svůj produkt? (mimo HPP a spol)
- Jak probíhá vaše volba programovacího jazyka?
- Jak testujete svůj SW?

Proč sofwarové inženýství (1 / 3)

- $\Rightarrow$ Nejste první ani poslední, kdo vyvíjel/bude vyvíjet nějaký produkt
- Od počátku počítačů (2. světová válka) potřeba vyvíjet software
- Lidé byli zvyklí řídit poměrně komplikované fyzické procesy
  - Výroba složitých strojů (letadla, automobily)
  - Organizace státu
  - Mezinárodní obchod a logistika
- Se software přichází něco nového, nehmatatelného
- Kompletně nové odvětví, kde nemusí platit zavedené pořádky

Proč sofwarové inženýství (2 / 3)

- Kolik (bezchybných) řádků kódu zvládnete napsat za den?
- (Brooks 1975): „Programátoři jsou schopni napsat cca 1000 řádků odladěného kódu za rok“
- Komplexnost projektů výrazně zvyšuje náročnost vývoje
  - Nepřenositelná zkušenost z malých projektů
- Velké projekty mají vysokou režii
  - Plánování, modularizace, definice rozhraní, dělba práce
  - Specifikace, testování, kompletace
- Projekt trval 20 lidem 2.5 roku, jak dlouho bude trvat 40 lidem?
- (Brooks 1975): „Adding manpower to a late software project makes it later.“

Proč sofwarové inženýství (3 / 3)

- Postupný přechod funkcionalit z HW do SW
  - Jak dnes vypadá instrukční sada a architektura dnešních procesorů?
  - Je levnější upravit SW nebo HW?
- První velké SW projekty měly obrovské zpoždění a zvýšené náklady
- Potřeba proces vývoje efektivně řídit a plánovat
  - Vznik SW inženýrství
  - Aplikování známých metod pro nové odvětví
  - Odhalení chyb, poučení se z nich, aplikování do nových projektů
- Standardizace některých postupů
- Doteď jsme mluvili vágně, pojďme si věci definovat

Software

- Co to vlastně je software? Software $\neq$ pouze program
- Komplexní balík programu,(systémové a uživatelské) dokumentace a konfiguračních souborů potřebných pro bezproblémový chod programu
  - SW bez zaškolených uživatelů (z dokumentace) nemusí fungovat správně
  - Dobrý, ale špatně nakonfigurovaný SW může být také nefunkční
- Základní typy prodávaného SW (produkt)
  - Generický produkt – samostatně vyvíjený produkt, výrobce umísťuje na trh
    - „krabicové programy“, operační systémy, grafické editory, kancelářské balíky, střižny, ...
  - SW „na míru“ – určitý zákazník se podílí na vývoji (specifikace), vlastní i práva
    - ERP, e-shop, CMS, Identity management
- Rozdělení dnes není striktní $\Rightarrow$ obecné řešení „ohnuté“ na míru zákazníkovi
  - Zákazník určuje pouze konkrétní podobu jedné instance generického SW
  - Práva ošetřena smluvně

Softwarové inženýrství (SI)

- Inženýrství = Aplikace teorie v praxi.
  - Zvolení ověřených metod a nástrojů pro splnění cíle.
  - Snaha nalézt řešení problémů, i když známá teorie selhává či není k dispozici.
  - Nutnost řešení i za různých organizačních podmínek (finance, lidské zdroje, právo)
- Softwarové inženýrství = aplikace inženýrských metod na všechny složky vývoje SW
  - Specifikace SW
  - Vývoj
  - Testování
  - Projektový management
  - Údržba
  - Integrace s ostatními systémy, ...

SI vs. Informatika

- Informatika se zabývá teorií a metodami, na kterých jsou založené počítačové a softwarové systémy
  - Návrhy algoritmů (složitost, výčíslitelnost, . . .)
  - Princip fungování počítačů (výpočetní modely, architektura PC)
  - Skládání a běh programů (překladače, architektury, VM)
- Softwarové inženýrství řeší praktické problémy při tvorbě software
  - Znalost informatiky je pro SI potřeba
  - Někdy používání neexaktních ad-hoc metod
  - Nemusí být dostupné přímé srovnání metod (obtížné a drahé)

Softwarový proces

- = soubor aktivit a procesů potřebných k dokončení softwarového produktu. Složen z 4 částí:

1. **Specifikace** – zákazník a vývojáři definují, jak má výsledný produkt za jakých podmínek fungovat
2. **Vývoj** – samotný návrh a programování (Je to „to hlavní“?)
3. **Validace** – ověření, zda vyvinutý SW splňuje specifikaci
4. **Evoluce** – úpravy SW měnícím se potřebám zákazníka/trhu

- Různé projekty vyžadují výše zmíněné body v různé kvalitě
  - Příklady (řízení letového provozu, eshop, osobní projekt, . . .)

Modely softwarového procesu

- Aplikací osvědčených metod na podobné projekty dojdeme k podobnému cíli
  - Architektura budov, design a výroba aut, sport, ...
  - Architektura aplikací
- $\Rightarrow$ Základní modely (paradigmata) softwarového procesu

1. **Vodopádový model** – dříve uvedené části striktně (časově) odlišuje
   - Po dokončení jedné části se „uzamkne“ a pokračuje se další
2. **Evoluční model** – Specifikace, vývoj a validace se překrývají
   - Rychlé vyvinutí prototypu, následné úpravy. Na konec možná reimplementace
3. **Formální transformace** – vychází se z matematické formální specifikace
   - Specifikace se postupně transformuje do funkčního programu
4. **Znovupoužití existujících komponent** – integrace existujících komponent
5. **iterativní modely** – inkrementální vývoj, spirálové modely

Modely SW procesu – výběr

- Tak který je tedy ten nejlepší?
- $\Rightarrow$ obecně žádný
- Pro různé typy a velikosti vyvíjeného SW je potřeba volit individuálně
- Může být vhodné dané modely kombinovat – pro různé části SW, různé modely
  - Např. Dnes populární architektura aplikací založená na mikroslužbách
- Dnes také populární agilní metody (extrémní programování, SCRUM, Kanban, . . .)

Časové plány modelů

- Jakou porci času (a peněz) zaberou jednotlivé fáze daných modelů?
- Vodopádový model
  - Specifikace $\approx$ 15 %
  - Design $\approx$ 25 %
  - Vývoj $\approx$ 20 %
  - Integrace a testování $\approx$ 40 %
- Iterativní vývoj – fáze se překrývají, nelze měřit samostatně
  - Specifikace $\approx$ 10 %
  - Iterativní vývoj $\approx$ 60 %
  - Testování $\approx$ 30 %
- Znovupoužití komponent
  - Specifikace $\approx$ 15 %
  - Vývoj $\approx$ 35 %
  - Integrace a testování $\approx$ 50 %
- Poměr vývoje a evoluce (údržba)
  - 1:4 $\Rightarrow$ Vývojem systém nekončí. Evoluce a údržba zabere mnohem více

Dobrý software

- Abychom SW označili jako *dobrý*, nestačí, aby jen „dělal co má“
- Důležité je i „jak to dělá“ a také jakým způsobem je navržen
- **Bezpečnost** – neobsahuje bezpečnostní chyby? Jak manipuluje s našimi daty?
- **Efektivita** – Sw by neměl plýtvat prostředky (HW, čas, paměť). Šetří peníze
- **Udržitelnost** – Jak náročné je zanášet do SW změny? Velmi důležité
- **Použitelnost** – Pracuje se se SW snadno? Zaučí se uživatelé rychle?
  - Odezva UI
- **Spolehlivost** – Pády SW, ztráta dat, „zamrzání“

Doménová znalost

- ? Dobrý programátor = dobrý softwarový inženýr ?
  - Neplatí to
- Softwarový inženýr musí mít základní znalosti domény, pro kterou navrhuje SW
- Čím je „blíže ke kódu“, tím méně je potřeba
- Př.: Zákazník chce naprogramovat systém řízení kotelny na tuhá paliva. Dodá požadavek:
  Regulátor bude ovládat servomotor trojcestného směšovacího ventilu podle ekvitermní křivky, dále bude řídit oběhové čerpadlo bojleru a spínat relé topné spirály, pokud není dostatečná teplota v akumulačních nádržích a kotel v provozu a je sepnuté HDO.
  - Věděli byste, co máte dělat?
  - SI musí porozumět pojmům, navrhnout specifikaci SW a jeho desing a předat práci programátorům

Známe softwarově inženýrské rady (a mýty?)

- Paretův princip (80/20)
  - *Za* 20 % *času jsme schopni udělat* 80 % *vývoje*
  - Zbylých potřebných 20 % Nám zabere 80 % času
  - $\Rightarrow$ Máte už 80 % bakalářky?
  - Platí to?
- *Nehrabej do něčeho, co funguje a nikdo si nestěžuje*
  - Při chtěné změně vstupuje do rozhodovací funkce i ochota programátorů a riziko chyby
- Nezačínej aktualizaci systémů v pátek odpoledne
- Dokumentace je zbytečná zdržovačka
- SI a modely vývoje jsou zbytečné, zvládneme to „bouřlivým vývojem“

Sociálně technické systémy

- SI není disciplína postavená na „zelené louce“
  - Aplikování dřívějších poznatků do nového oboru
- Co to je systém? Pouze suma jeho částí? Něco navíc?
- $\Rightarrow$ *Kolekce vzájemně souvisejících komponent, které spolupracují k dosažení konkrétního cíle*
- Příklady:
  - Volební systém
  - Systém pro řízení letového provozu
  - Kávovar
  - Sluchátka
  - Auto
  - ...

Sociálně technické systémy

- Zaměříme se na systémy, kde hraje roli software

1. Technické počítačové systémy
   - Zahrnují HW a SW, ale neobsahují žádné procesy (lidé, právo, byrokracie)
   - Slouží ke splnění svého cíle, nestarají se o „big picture“
   - Obrázkový editor, mobilní telefon, klávesnice, ...
2. Sociálně technické systémy
   - Většinou obsahují několik technických počítačových systémů
   - Zahrnují procesní záležitosti, lidské síly (operátory), právo, byrokracie.
   - Fotobanka, ochrana firemních dat na zaměstnaneckých telefonech, zpracování výplat ve firmě

- Softwarové inženýrství není jen o programování
  - Musí reflektovat kontext, ve kterém je SW vyvíjen
  - Znát procesní potřeby zákazníka, znalost prostředí

Vlastnosti sociálně technických systémů

1. Mají *emergentní(skryté,vyvstalé) vlastnosti*, které nesouvisí s žádnou konkrétní částí
   - Reflektují vztahy mezi komponentami
   - Často „se objeví“ až když je systém kompletně sestaven
2. Mohou být nedeterministické – při stejném vstupu nemusí dát stejný výstup
   - Zahrnují lidi, mohou se chovat odlišně (stres, únava, nálada, . . .)
3. Splnění cíle systému nemusí být čistě v jeho rukou
   - Mylná interpretace cíle lidmi (osobní „sabotáž“, neochota učit se novým věcem, . . .)
   - Změna směřování organizace $\Rightarrow$ změna cíle

- Systém bývá často složen z dílčích *subsystémů*
  - Selhání subsystému může způsobit selhání celého systému
  - SW bývá často v roli pružného „lepidla“ mezi subsystémy
  - Snadněji změníte SW než drahý HW, lidské chování, . . .

Emergentní vlastnosti systémů

- = vlastnosti systému jako celku, nesouvisí (pouze) z jednou komponentou
  - Vyvstanou ze vzájemných vztahů mezi komponentami
  - Nejdou vyčíst z jednotlivých vlastností jednotlivých komponent

Základní dělení:

1. **Funkční emergentní vlastnosti** – objeví se, až celý systém začne pracovat na splnění cíle
   - Např.: Mobil po sestavení nabootuje a umožní uskutečnit hovor
2. **Nefunkcionální emergentní vlastnosti** – chování systému v prostředí působnosti
   - Spolehlivost (HW, SW)
   - Výkon/efektivita
   - Bezpečnost
   - Opravitelnost
   - Údržba/správa

Systémové inženýrství

- Tradiční($\approx$ staletí) inženýrská disciplína zabývající se specifikací, designem, implementací, ověřováním, nasazování a údržbou sociálně technických systémů
- Nutná mezioborová znalost – ne pouze HW/SW – interakce s uživateli, s prostředím, normami, kulturou, ...
- Odrazový můstek pro Sowftwarové inženýrství
  Základní fáze:
  1. Specifikace požadavků
  2. Návrh systému
  3. Vývoj subsystémů
  4. Integrace
  5. Instalace
  6. Evoluce
  7. Ukončení provozu

Systémové inženýrství

- Boj s fyzikálními limity
- Velmi častou komplikací je omezený rozpočet
- Omezené možnosti zpětného přepracování částí systému
  - Jakmile je již systém instalován, nemusí jít některé části změnit (např. drahé)
  - Je potřeba velmi promyslet samotný návrh $\Rightarrow$ výhoda flexibilního SW
- Mezioborová spolupráce – může docházet k nedorozuměním
  - Stejným pojmům mohou experti z různých disciplín říkat jinak
  - Fyzikální jednotky (pád Mars Climate Orbiter)
- Co vše musíme znát pro vytvoření systému pro řízení letového provozu?
  - Inženýrství SW, elektroniky a elektřiny, mechaniky
  - Statika, architektura, návrh uživatelského rozhraní, ...

Specifikace požadavků

- Zákazník/dodavatel systému musí přesně specifikovat cíle a vlastnosti celého systému
- Není žádoucí popsat všechny požadavky a vlastnosti do stejného detailu
  - Snadnější orientace, vysvětlení principů

1. **Abstraktní popis funkčních požadavků** – popis základních funkcí na abstraktní úrovní
   - Detaily budou řešeny na úrovni subsystémů
   - Např.: *Telefon umožní uskutečnit hovor a psát SMS.*
2. **Definice vlastností systému** – popis nefunkcionálních emergentních vlastností
   - Jak má být systém spolehlivý? Jak má být bezpečný?
   - Tyto vlastnosti by měly být zohledněny v každém subsystému
   - Např.: *Telefon vydrží alespoň 100 hodin v pohotovostním režimu*
3. **Popis, jak se systém nemá chovat** – vymezení hranic z obou stran
   - Může být vhodnější popsat, co systém dělat nesmí
   - Např.: *Telefon neumožní číst a odesílat zprávy bez zadání hesla*

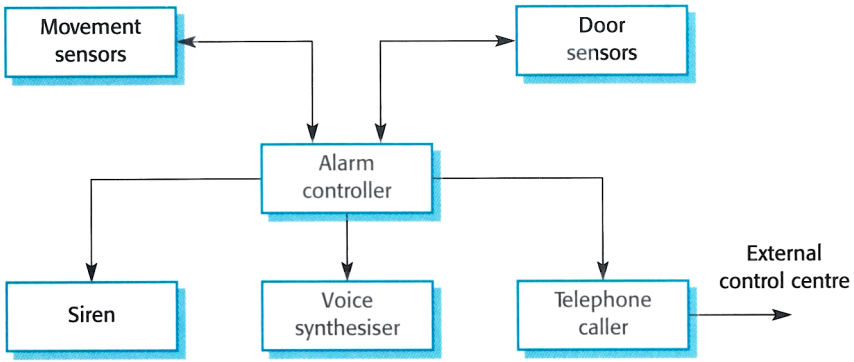
Návrh systému

1. **Rozdělení požadavků** – rozdělení požadavků do skupin
   - Většinou existuje více možností rozdělení
2. **Identifikace subsystémů** – které subsystémy by mohly splnit které požadavky?
   - Zohlednění jak technické, tak organizační faktory
3. **Přiřazení požadavků subsystémům**
   - Přizpůsobení požadavkům externím(nezměnitelným) subsystémům
4. **Specifikace funkcionality subsystémů** – co který systém přesně dělá
   - Je-li subsystém SW⇒ začátek softwarového procesu
5. **Definice rozhraní mezi subsystémy** – jak spolu budou systémy interagovat
   - Možný paralelní vývoj subsystémů
   - Snadnější budoucí výměna subsystémů
6. Jedna fáze ovlivňuje druhou (i zpětně) ⇒ překrývání fází ⇒ spirálový vývoj
   - Možná zpětná modifikace „předchozích“

Modelování systémů

- Při specifikaci požadavků a návrhu systému vytváříme abstraktní model systému
  - $\Rightarrow$ oprostíme se od technických detailů, zaměření na to hlavní
  - Různá míra abstrakce, konkrétnosti – „zoom“ na systém
- Používá se blokový diagram komponent
  - Subsystémy obdélníky s názvem
  - Vztahy mezi nimi (oboustrannými) šipkami
  - Vztahy pojmenovány např.: *Používá, spíná, . . .*
- Blokový diagram je vhodné doplnit tabulkou
  - Pojmenování a krátký popis jednotlivých subsystémů
- Pro každý subsystém může být opět blokový diagram
  - Různá míra abstrakce
  - Od drobných diagramů pro malé systémy po složité diagramy velkých systémů

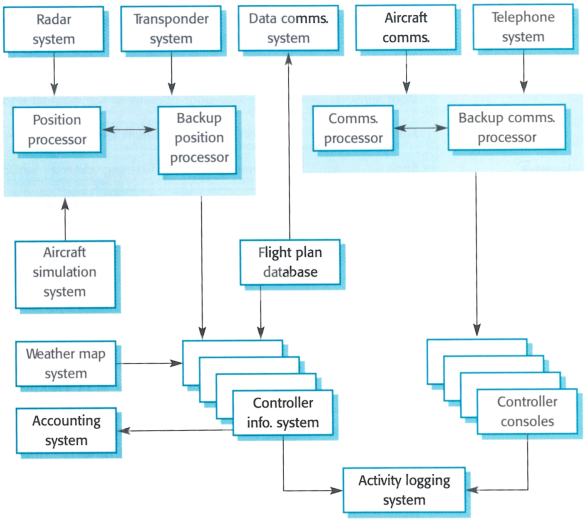
Modelování systému – diagram



Modelování systému – tabulka subsystémů

Sub-system	Description
Movement sensors	Detects movement in the rooms monitored by the system
Door sensors	Detects door opening in the external doors of the building
Alarm controller	Controls the operation of the system
Siren	Emits an audible warning when an intruder is suspected
Voice synthesiser	Synthesises a voice message giving the location of the suspected intruder
Telephone caller	Makes external calls to notify security, the police, etc.

Modelování systému – složitější systém



Vývoj/použití subsystémů

- Pro jednotlivé subsystémy můžeme proces opakovat znovu
  - Výchozí bod = požadavky na subsystém
- Budeme každý subsystém navrhovat „od podlahy“
  - Co tahle využít stávající řešení – *Components of the shelf (COTS)*
  - Výhody? Nevýhody?
- Použití COTS může výrazně zlevnit systém
  - Ovlivnění a snazší odhalení emergentních vlastností (hotové komponenty budou vyladěné, zkušenosti z jiných systémů)
- Komponenta OTS nemusí úplně splňovat požadavky
  - Zbytečná funkcionalita
  - Vendor lock-in
  - Životnost (morální), rozhraní

Integrace systému

- Kompletovat všechny komponenty systému zároveň? Nebo postupně?
- Většinou nemáme k dispozici hotové všechny subsystémy ve stejný čas
  - Obtížné načasování, opožděné dodávky
- Při postupné integraci se snadněji hledají chyby
  - Odladí se vždy jedna část a chyby s ní související
- Po sestavení je potřeba otestovat rozhraní subsystémů
- Odhalení chyby („kdo za ni může“) bývá složité
  - Je chyba ve specifikaci nebo v provedení?
  - Je řešení chyb součástí smlouvy?
- S využitím COTS je integrace složitější, větší důraz

Evoluce systému

- Životnost systémů od měsíců po desítky let
- Změna požadavků, nové skutečnosti, odladění chyb
  - Životnost operačního systému (bezpečnost), protokolů
- Každá změna by měla být pečlivě zvážena (Proč se dělá? Splňuje cíl?)
- Změna v jednom subsystému může nečekaně ovlivnit ostatní
  - I když dodržíme rozhraní
  - Např. výrazné zrychlení jednoho subsystému $\Rightarrow$ přesun úzkého hrdla jinam
- Problém špatné dokumentace systému – spousta věcí nedokumentována, zapomenuta (proč je něco takové, jaké to je)
- Čím starší, tím hůře se změny dělají (legacy systémy)
- Čím více změn, tím se systém znepřehledňuje

Vyřazení systému z provozu

- Každý systém by měl mít naplánovanou životnost
- Co se bude dít po jejím ukončení
- Demontáž, recyklace HW komponent (kdo to bude platit?)
- Co se bude dít s daty? Zničení? Archivace
- Často opomíjená část
- Odhady životnosti mohou být chybné
  - Systém je pořád potřeba a funguje
  - Návrh systému s něčím nepočítal
  - Příliš krátká životnost (EET)
  - Příliš dlouhá životnost (Opportunity Rover – 90sol vs. 5110sol)
  - Prodloužení podpory OS (Windows XP na bankomatech)

Doporučená literatura

- Ian Sommerville. Software Engineering (10th Edition). Pearson, 2015. ISBN: 978-93-325-8269-9.
  - Chapter 1 – Introduction to Software engineering
- Ian Sommerville. Software Engineering (7th Edition). Pearson, 2004. ISBN: 0-321-21026-3.
  - Chapter 2 – Socio-technical systems