

# Poznámky k předmětu **Softwarové inženýrství pro praxi**

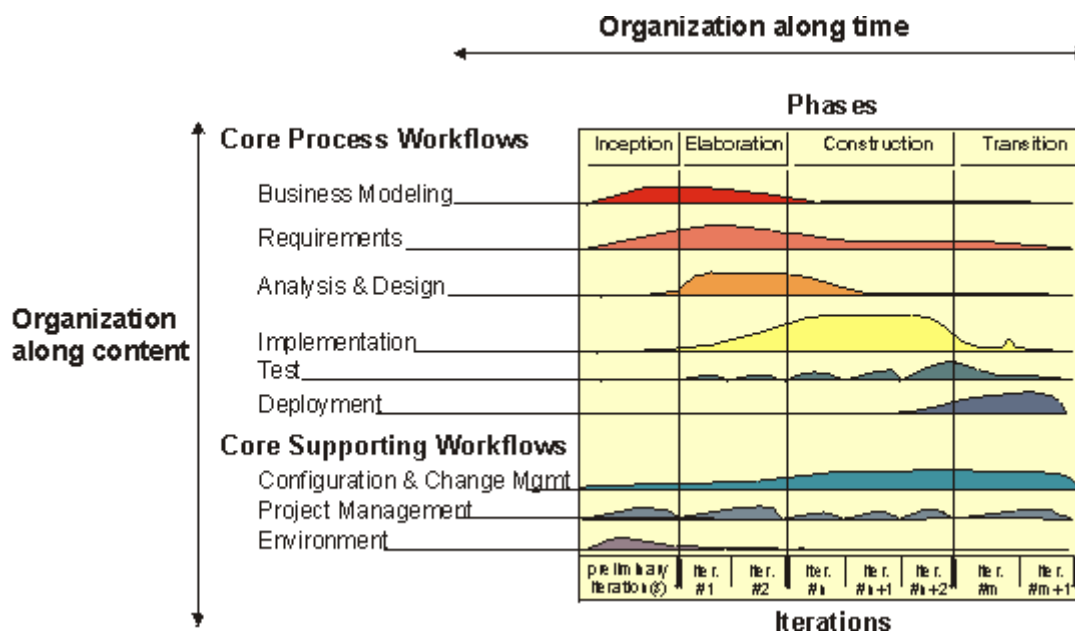
- zpracováno na základě přednášek a slajdů Tomáše Krátkého ([Profinit s.r.o.](http://www.profinet.eu)), © 2009 Profinit
  - <http://www.profinet.eu/cz/podpora-univerzit/univerzitni-vyuka/swi129>
- svými zápisky doplnil a dohromady sestavil: [Bohumír Zámečník](#) v akademickém roce 2009/2010

## 1. Introduction

- software process



- Stages of the SDLC: Assess Needs, Design Specifications, Design / Develop Software, Implement Systems, Support Operations, Evaluate Performance (<http://www.ains-inc.com/sdlc-support.html>)



- zdroj: Rational Unified Process: Best Practices for Software Development Teams  
[http://www.augustana.ab.ca/~mohrj/courses/2000.winter/csc220/papers/rup\\_best\\_practices/rup\\_bestpractices.html](http://www.augustana.ab.ca/~mohrj/courses/2000.winter/csc220/papers/rup_best_practices/rup_bestpractices.html)
  - přechody mezi časovými fázemi: LCO, LCA, IOC
- modely životního cyklu software
  - návaznost primárních činností
  - waterfall – jednotlivé části striktně sekvenčně na jeden pokus, hodí se spíš pro fázi údržby
  - iterativní, evoluční – dynamická reakce na změny, používají se častěji
- COCOMO (CONstructive COst MOdel) – metodologie, jak odhadovat cenu projektu
  - důležité body při vývoji
    - LCO (Life Cycle Objectives) – víme, co zákazník chce
    - LCA (Life Cycle Architecture) – víme, jak to udělat
    - IOC (Initial Operational Capability) – máme nasazeno
- obecné materiály
  - monografie

- Pressman R. **Software Engineering: A Practitioner's Approach**. 5th ed., McGraw-Hill, 2001.
- Dorfman, M. and Thayer, R. **Software Engineering**. IEEE Computer Society Press, 2000.
- Sommerville, I. **Software Engineering**. 8th ed., Addison Wesley, 2006.
- Spolsky, J. **Joel on Software: And on Diverse and Occasionally Related Matters That Will Prove of Interest to Software Developers, Designers, and ...**. Apress.2004 (dobrodružnější četba)
- žurnály
  - [CrossTalk: The Journal of Defense Software Engineering](#)
  - [ACM SIGSOFT Software Engineering Notes](#)
  - [IEEE Software](#)
- webové stránky
  - [Software Engineering Institute](#)
  - [Construx](#)
  - [Process Impact](#)
  - [Pressman Software Engineering Resources](#)
  - [SWEBOK – Guide to the Software Engineering Body of Knowledge](#)
    - mapa softwarového inženýrství – mnoho odkazů na literaturu

## 2. Requirements Engineering

- stručný přehled tématu
  - schematický pohled
    - (Software System) Requirements Engineering
      - Elicitation – (schůzky, jednání, připomínkování dokumentů, pozorování uživatelů)
      - Analysis – (přemýšlení, vymýšlení, debaty, poznámky, ...)
      - Specification – (dekompozice, psaní, používání notace)
      - Verification – (čtení textu, schůzky, jednání, promítání GUI, velké bitvy o rozsah)
    - to vše i několikrát, promixované v čase, lidech, zaměření, ...
  - cíl: dostat se do stavu, kdy:
    - my rozumíme, co chce uživatel
    - uživatel rozumí, co my nabízíme
  - rozsah
    - všechny závazky!!!
      - zákazník potenciálně chce vše, co není explicitně „NE“
      - dodavatel většinou dělá jen to, co je explicitně „ANO“
      - pozor na šedou zónu
    - kromě specifikace funkcí je také třeba specifikovat:
      - použité metodologie, integrace s dalšími systémy, bezpečnost, lokalizace
      - rozhraní – GUI, OS, hardware, databáze, aplikační software, knihovny, ...
      - výkon – přemýšlet už i o architektuře, být konkrétní
      - právní aspekty – normy, bezpečnostní audity, certifikace, ochrana osobních údajů
  - typy požadavků
    - požadavky na vlastní funkce
    - požadavky na rozhraní
    - nefunkční požadavky
    - další požadavky
  - kontext softwarového procesu
    - primární aktivita
    - graf (kulatý) – fáze Assess Needs
    - graf (hraný) – řádek Requirements Capture
    - SWEBOK
- zásadní otázky
  - co má být výsledek analýzy?
  - obsah?
  - forma?

- logická a fyzická dekompozice?
- míra detailu?
- pracnost?
- kalendářní čas?
- počet lidí?
- jak mezi paralelně pracujícími lidmi dekomponovat práci?
  - de facto už ovlivňují výsledek analýzy její předčasnou dekompozicí
- kdy mohu už začít navrhovat architekturu?
- kdy mohu už začít konstruovat?
- jak má být analýza/specifikace rozprostřena v čase od psaní nabídky až po údržbu systému?
- jaké jsou rozdíly mezi specifikací z nuly vs. specifikací změnových řízení během údržby systému
- jaký je vlastně vztah mezi specifikací a architekturou (co vs. jak)?
  - pouze architekt může říci, co lze nebo nelze implementovat
- jak poznám, že na straně zákazníka se mi věnují ti správní lidé?
- jaké vlastnosti má mít dobrý analytik?
- jak ověřím, že specifikace je specifikace toho, co se skutečně chce, resp. potřebuje?
- je rozumné bát se zeptat?
- je rozumné nechat si schválit něco, o čem mám sám vnitřní pochyby?
- jak detekovat nárůst rozsahu?
- fenomén „gold-plating“?
- a mnoho dalších ...
- poznatky z praxe
  - pracnost: 10-30 %
  - pracnost při údržbě: 20 %
  - rozložení v čase je podle "knih"
  - osobní předpoklady – ne ghetto analytiků – je potřeba silné vazby mezi architekty a analytiky
  - forma nesmí zastínit obsah
  - obsah musí být „komplexní“
  - rozsah musí být analýzou dost popsán
  - udržovatelnost je zásadní věc
  - čistý princip
  - SDLC hraje roli
  - rozložení výdajů pracnosti
  - nutnost mít odvahu se poučit
  - model GUI a strukturovaný text funguje
  - template a checklist funguje
  - všechny typy požadavků jsou třeba
  - naměřená data jsou třeba (obrazovky, ...)
  - život si prosadí, co je skutečně třeba (otázkou je, jak bolestně)
  - co se změnou požadavků?
    - změna nebo odložené pochopení
    - změna nebo větší míra detailu
    - pozitivní a negativní vymezení
    - boj o rozsah
- slepé uličky
  - nedělat analýzu
  - nemyslet na architekturu
  - dělat pouze katalog či use cases
  - ignorovat jiné než funkční požadavky
  - nepoznat, kdo je skutečně důležitý stakeholder
  - nevěřit vlastní rozvaze a intuici
  - neptat se na věci, které „nechci“ slyšet
  - a mnoho dalších ...
- zajímavá témata

- textová specifikace – styl
  - viktoriánský (slohovka) – ne moc použitelný
  - strukturovaný (viz SVZ) – běžně použitelný
  - izolovaný (katalog) – použitelný jako doplněk
- dekompozice požadavků
  - kolem čeho?
    - Data-centric
    - Function-centric
    - Feature-centric (viz datové schránky) ← do LCA
    - Use case-centric
    - Aspect-centric
    - Architecture (imposed/proposed ← při vývoji)-centric
    - Change request-centric (viz IS VZ US) ← po IOC (údržba)
  - jak?
    - UML-centric
    - Text-centric
- model GUI
  - funguje skoro cokoli!
    - PowerPoint (viz eSIPO)
    - Excel (viz datové schránky)
    - HTML (potenciálně základ GUI systému)
    - speciální jazyky pro tvorbu modelu GUI
  - ne vždy je třeba, ne vždy dává smysl
  - pomáhá pochopení systému
- UML
  - prostředek pro reprezentaci vyvíjeného SW – na úrovni analýzy, návrhu a částečně i realizace
  - nutné znát (problém u zákazníka)
  - nemá smysl vymýšlet něco jiného
  - ne vždy je použitelný
- Use case
  - scénáře
  - dobré jako doplněk (viz datové schránky)
  - nelze použít jako základ pro specifikaci
  - někdy jen útěk před složitostí!
- ilustrace
  - SVZ – jednoduchá strukturovaná specifikace
  - IS VZ US – dekompozice dle change requests, specifikace změnového řízení 4907
  - eSIPO – PowerPoint model GUI
  - Datové schránky – funkční uživatelská specifikace, využití Use cases, XLS model GUI
- goodies, literatura
  - články
    - [When Telepathy Won't Do: Requirements Engineering Key Practices](http://www.processimpact.com/articles/telepathy.html)
      - <http://www.processimpact.com/articles/telepathy.html>
    - [Karl Wiegers Describes 10 Requirements Traps to Avoid](http://www.processimpact.com/articles/reqtraps.html)
      - <http://www.processimpact.com/articles/reqtraps.html>
    - WritingEffectiveSRS.pdf – **Writing Effective Natural Language Requirements Specifications**
    - BeCarefulWithUseCases.pdf – **Be Careful With “Use Cases”**
  - knihy
    - Wieger K. **Software Requirements: Practical techniques for gathering and managing requirements throughout the product development cycle.** Microsoft Press, 1999. resp. 2nd ed.
    - Ed Yourdon: [Just Enough Structured Analysis](http://yourdon.com/strucanalysis/wiki/) – klasická kniha, wiki formát
      - <http://yourdon.com/strucanalysis/wiki/>
  - checklists
    - CxCheck\_Requirementst.txt

- Impact\_analysis\_checklist.doc
- Requirements\_review\_checklist.doc
- Use\_case\_checklist.doc
- templates
  - srs\_template.doc
  - use\_case\_template.doc
  - vision\_and\_scope\_template.doc
  - SAFE\_BusinessRequirements.doc
  - SAFE\_SystemRequirements.doc

### 3. Software architecture & design

- stručný přehled tématu
  - kontext softwarového procesu
    - primární činnost
    - fáze Design Specifications
    - řádek Analysis & Design
    - SWEBOK – Software Design
      - Software Design Fundamentals
      - Key Issues in Software Design
      - Software Structure and Architecture
      - Software Design Quality Analysis and Evaluation
      - Software Design Notations
      - Software Design Strategies and Methods
- zásadní otázky
  - co je to vlastně architektura? struktura
    - typy architektury?
      - Software architecture
      - System architecture
      - Process architecture
      - Enterprise architecture (architektura celého podniku), ...
    - pohledy: Business Processes → data → aplikace → technologie (a opačným směrem)
    - jakou roli hraje architektura na projektu?
    - architektura vs. design
      - co je to architektura, co návrh?
        - architektura – obecnější
        - design – konkrétnější
        - dohromady:
          - návod pro implementaci
          - základ pro plánování
          - soubor myšlenek a pravidel
      - kde je hranice mezi nimi? plynulý přechod
      - architektonicky významné rozhodnutí?
      - jak poznám, co je důležité?
      - jaký je průnik množin Architecture a Design?
      - „Architecture is about the important stuff. Whatever that is ...“ (Martin Fowler, Who needs an Architect?)
  - Design
    - základní koncepty
      - dekompozice (decomposition)
      - abstrakce (abstraction)
      - zapouzdření (encapsulation)
      - koheze (cohesion (high)) – jedna část dělá jeden úkol
      - vazby (coupling (low))

- základní pojmy
  - abstraktní datový typ (ADT)
  - typ (type) – množina instancí se společnými vlastnosti a chováním
  - třída (class) – implementace typu
  - instance
  - objekt (object) – implementace instance
  - modul (module)
  - a mnoho dalších:
    - composition, association, delegation, inheritance, implementation inheritance, multiple inheritance, dynamic binding, static x dynamic type, information hiding, encapsulation, ad hoc polymorphism, component, OO framework, class category, members (data m., method m.), behavior (external), messages, reflection, ...
- dobrý návrh
  - Program to interface, not implementation!
  - Favor object composition over class inheritance!
  - Keep it DRY, shy and tell the other guy! (Don't Repeat Yourself)
- architektura
  - architecture needs, stakeholders – *Stakeholder* → *Concern*
    - architektura interguje požadavky od všech stakeholderů
    - stakeholder = ten, kdo má nějaký zájem na daném softwaru
    - Customer
      - schedule and budget estimation
      - feasibility and risk assessment
      - requirements traceability
      - progress tracking
    - User
      - consistency with requirements and usage scenarios
      - future requirements growth accomodation
      - performance, reliability, interoperability, etc.
    - Architect and System Engineer
      - requirements traceability
      - support of tradeoff analyses
      - completeness, consistency of architecture
    - Developer
      - sufficient detail for design
      - reference for selecting / assembling components
      - maintain interoperability with existing systems
    - Maintainer (vývojář programující nové věci, administrátor, ...)
      - guidance on software modification
      - guidance on architecture evolution
      - maintain interoperability with existing systems
  - dokumentace architektury – 4+1 view model
    - Logical View, Development View, Process View, Physical View, Scenarios
    - Logical View → (Development View, Process View) → Physical View
    - Logical View – funkcionalita pro koncové uživatele (Class diagram, Communication d., Sequence d.)
    - Development (Implementation) View
      - pohled programátora, konkrétní jazyky, platformy, knihovny
      - Component diagram, Package diagram
    - Process view
      - chování systému zaběhu, systémové procesy (procesy na úrovni OS) a jejich komunikace
      - konkurence, distribuovanost, integrace, výkon, škálovatelnost; Activity diagram
    - Physical view
      - pohled systémového inženýra či administrátora
      - topologie a komunikace komponent na fyzické úrovni (počítače, síťové prvky, ...)

- Deployment diagram
  - Scenarios
    - sekvence interakcí mezi objekty a mezi procesy
    - výchozí bod pro testování prototypu architektury
    - identifikace elementů architektury, prostředek pro validaci architektury
    - Use case diagram
  - vliv kontextu na architekturu
    - databázový systém / subsystem
    - web systém / subsystem
    - (tlustý) klient systém / subsystem
    - objektově orientovaný systém / subsystem
    - data warehouse systém
    - integrační systém / subsystem, ...
- zajímavá témata
  - objektově orientované systémy
    - výhody
      - Reusability
      - Maintainability
      - Portability
      - Customizability
      - Extensibility
    - techniky
      - Data encapsulation
      - Data abstraction
      - Interface and code sharing
      - Polymorphism
      - Design sharing
  - Design patterns
    - katalog
      - základní návrhové vzory od GOF (Group of Four)
      - lze kombinovat prakticky do nekonečna
    - význam – znovupoužitelnost, společný jazyk, ...
    - pozor: na počáteční nadšení, na nadbytečné užívání patterns
      - nejpoužívanější pattern? Proxy!
  - Architectural styles
    - Pipes and filters
    - Event driven architecture
    - Layered architecture – interní dekompozice (např. TCP/IP)
    - Multi-tier architecture – fyzická dekompozice (např. webový server – aplikační server – databáze)
    - Model-View-Controller (MVC)
    - Repositories
    - „Table driven” interpreters
    - Big ball of mud :)
    - a mnoho dalších ...
  - OO frameworks
    - znovupoužitelný návrh pro SW systém
    - podpora (základna) při vývoji jiných SW aplikací
    - diktuje architekturu systému – určuje, jak dekomponovat systém a jak budou jeho jednotlivé části komunikovat
    - Frozen spots a hot spots (abstraktní třídy, anotace)
  - integrace
    - velmi zajímavé a časté téma prakticky u každého většího projektu
    - často spojené s tematikou enterprise architektury
    - často velmi netechnologické (procesy, entity)

- uživí se zde mnoho buzzwords (EAI, SOA, MOM, ...)
- obvykle velmi problematické (odpovědnost a peníze chybí, neochota, ...)
- základní koncepty – nic zásadně odlišného snad ani není
  - File transfer
  - Shared database
  - Remote procedure call
  - Messaging – např. Enterprise bus (Composite services)
- Goodies – templates, checklists
  - CxCheck\_HighLevelDesign.txt
  - CxCheck\_HighQualityModules.txt
  - CxCheck\_SwArchitecture.txt
- Doporučená literatura
  - 01-Article original de Parnas.pdf
  - HowWhyFakeRationalDesignProcess.pdf
  - intro\_softarch.pdf
  - MIL-STD-498\_InterfaceDesignDescription.doc
  - MIL-STD-498\_InterfaceReqsSpecification.doc
  - MIL-STD-498\_SwDesignDescription.doc
  - MIL-STD-498\_SysSubsysDesignDescription.doc
  - MIL-STD-498\_SysSubsysSpecification.doc
  - Profinit\_SwArchitectureOverview.pdf
  - SwSystemArchitectureDef.pdf
  - WhoNeedsArchitect.pdf

## 4. Construction

- stručný přehled tématu
  - zařazení v rámci softwarového procesu
    - primární aktivita
    - součást fáze Design, Develop, Test Software
    - řádek Implementation
    - je třeba ctít přechody v grafu softwarového procesu → jinak hrozí problémy
  - derivace počtu nových řádek časem klesá
  - SWEBOK – Software Construction
    - Software Construction Fundamentals – Minimizing Complexity, Anticipating Change, Construction for Verification, Standards in Construction
    - Managing Construction – Construction Models, Construction Planning, Construction Measurement
    - Practical Considerations – Construction Design, Construction Languages, Coding, Construction Testing, Reuse, Construction Quality, Integration
- poznatky z praxe
  - programovat, když je jasné co (a jak)
  - ctít architekturu a design
    - pokud je na nich něco divné → diskuse/review architektury
  - používat vhodné nástroje a hotové věci → mít široký přehled
  - znát a chápat základní koncepty SW konstrukce
  - psát testy, aplikovat statickou analýzu kódu
  - pečovat o kód
  - vyhýbat se „rychlým, dočasným“ řešením
  - chápat ekonomiku projektu (firmy)
- zajímavá témata
  - Model/domain-driven development
    - co to znamená?
    - model-driven
      - vytvoření modelu → generování kódu (+synchronizace)

- Platform Independent Model (PIM )
  - export do konkrétního jazyka / platformy
- domain-driven
  - know-how o konkrétní doméně (např. pojišťovnictví)
  - model pojišťovny (obecné → konkrétní)
- jak se liší?
  - model-driven se snaží zachytit všechny low-level detaily
- jaká je podstata?
  - model (podstata) systému je pouze na jednom místě
- vazba na DSL?
- co je efektivní?
  - problém: v praxi lze těžko zachytit všechny detaily
  - vůči ručnímu programování málo efektivní
- jak správně aplikovat? pouze jako doplněk
- Test-driven development
  - co to znamená? něco víc než dogma: test, pak program
  - jaká je podstata?
    - kód, za **chvilí** unit testy; pokud nejdou, je program špatně
  - kdy je možný?
    - návrh programu, návrh testu → je testovatelný? → ne: oprava návrhu
  - jak na něj?
    - je třeba řešit simulaci okolního světa
    - např. v Profinitu: pokrytí / automatizace ~ 60-80 %, časová investice ~ 20 %
- Refactoring
  - co to znamená? zlepšení vnitřní struktury programu, tak aby byl efektivnější, lépe udržovatelný, pochopitelnější
  - kdy je možný?
  - jak na něj?
    - tip: explicitně naplánovaný refactoring
    - zobecněné řešení se hodí, pouze pokud se dá použít na další projekty
    - Projekt → Řešení → Zobecněné řešení (+iterace) → Komponenta
    - NE: Projekt → Komponenta, ani: Komponenta → Řešení
    - důležitý pro údržbu! stačí i 1,5-2 hodiny denně
- Self-documenting code
  - skutečný význam?
  - Literate programming?
    - moc nejde, např. pro zapouzdření
    - důležité jsou separátní projektové dokumenty
  - problémy: význam větších funkčních celků, pre/post conditions, invariants, inheritance, ...
    - → komentáře jsou potřeba, zachytí high-level informace
- Goodies – checklists
  - CxCheck\_Conditionals.txt
  - CxCheck\_ConstructingRoutine.txt
  - CxCheck\_ControlStructureIssues.txt
  - CxCheck\_DataCreation.txt
  - CxCheck\_DocumentingCode.txt
  - CxCheck\_HighQualityModules.txt
  - CxCheck\_HighQualityRoutines.txt
  - CxCheck\_Layout.txt
  - CxCheck\_Loops.txt
  - CxCheck\_NamingData.txt
  - CxCheck\_OrganizingCode.txt
  - CxCheck\_UnusualControlStructures.txt
  - CxCheck\_UsingData.txt

## 5. Testing

- stručný přehled tématu
  - co je Software testing?
    - praktická část Quality Assurance
    - zkoušení / simulace provozu SW
    - ustanovení důvěry v to, že SW dělá, co má dělat, a nedělá, co nemá dělat
    - analýza SW s cílem nalézt chyby a problémy
    - měření funkcionality a kvality SW
    - zhodnocení atributů a schopností SW, zda dosahují požadovaných či akceptovatelných výsledků
    - inspekce, stejně jako provádění testů kódu
    - shoda software se specifikací resp. ujasnění specifikace
      - po letech často důležitější zdroj informací o systému než samotná specifikace
    - Execution  $\subseteq$  Testing  $\subseteq$  Evaluation
  - trocha historie
    - **Testing objectives**
      - pomoci jasně popsat chování systému
      - nalézt defekty v požadavcích, designu, dokumentaci a kódu jak nejdříve je to možné
    - 1960s – demonstrace (ukázat fungování)
    - 1970s – detekce (hledat defekty)
    - 1990s – prevence (řízení kvality)
  - základní pojmy
    - **Test plan** (plán testů) – definuje strategii testů, vždy musí obsahovat:
      - **Test coverage** (co testovat, co netestovat)
      - **Test methods and tools** (jak testovat a pomocí jakých nástrojů)
      - **Test responsibilities** (odpovědnosti)
    - **Test case** (testovací případ) – množina podmínek, za kterých tester určí, zda aplikace či systém funguje korektně či nikoliv
    - **Test oracle** – mechanismus pro určení, zda SW prošel nebo neprošel určitým testem
      - požadavek – soulad se specifikací
      - regres – porovnání s předchozími výsledky
      - heuristika – možnost automatizace na mnohem vyšší úrovni
    - **Test script** – množina instrukcí (kroků), které budou provedeny na testovaném systému s cílem zjistit, zda systém funguje, jak je očekáváno
    - **Test data** (testovací data) – data speciálně identifikovaná pro využití v rámci testovacího případu
    - **Test report** (výsledky testu) – výsledek jednoho či více testů obsahující minimálně identifikaci testu a jednoznačný výsledek společně s komentářem, je-li třeba
  - základní principy
    - kompletní testování není možné
    - práce testerů je kreativní a náročná
    - testování je „řízeno“ riziky – testujeme. z čeho máme největší strach
    - analýza, plánování a návrh jsou důležité
    - motivace je důležitá
    - čas a zdroje jsou důležité
    - časování přípravy testů hraje velkou roli
    - měření a sledování „pokrytí“ je důležité
  - typologie testů
    - u dodavatele
      - unit testy
      - integrační testy
      - systémové testy
      - funkční testy
      - výkonové testy
      - bezpečnostní testy

- kvalifikační testy, ...
- u zákazníka
  - akceptační testy
  - uživatelské akceptační testy
  - operační testy, ...
- regresní testy
  - znovu testujeme systém po nějaké změně pomocí starých testů, jestli funguje, co fungovalo dřív
  - zafixovaná podmnožina unit / integračních / systémových testů
- kvalifikační testy
  - obdoba akceptačních testů, jen se odehrávají u dodavatele těsně před dodávkou
  - podmnožina běžných testů
- načasování testů aneb „V-model“ – testovací případy vymýšlíme už během plánování
- umístění testů v projektu – v různých prostředích
- SWEBOK
- začínáme testovat
  - jak (jednoduše) na testování?
    - základní strategie
      - začít se základními testy
        - testovat pomocí hodnot, které by měly projít, jinak to bude vážný problém
      - testovat nejdříve do šířky spíše než do hloubky
        - nejdříve zkontrolovat zběžně všechny části, než se zaměříme na detaily
      - pak se zaměřit na mocnější, komplexnější testy
        - jak program přežije základní otestování, je možné „přitvrdit“
    - vybrat vhodné hraniční podmínky
      - testů je příliš mnoho, je nutné vybrat vhodnou strategii pro jejich výběr
    - dělat exploratory testing („proklikání“ aplikace)
      - každý týden se snažit provést nějaké nové testy
  - základní myšlenky testů
    - **Boundary and Equivalence Analysis**
      - testů je příliš mnoho → Partitioning (rozdělení testů do tříd ekvivalence)
        - např. mobilní telefony, tiskárny, skenery
      - provádění více testů ze stejné skupiny je redundantní
      - volba nejvhodnějšího reprezentanta skupiny (ten s max. pravděpodobností odhalení chyby)
      - hraniční testovací případy jsou obvykle velmi mocné a identifikují často chyby
    - **Black box vs. White box**
      - Black box
        - testujeme oproti „rozhraní“
        - nezajímá nás implementace
        - robustnější (není nutné často upravovat)
      - White box
        - strukturální testy
        - přihlížíme k implementaci
        - křehčí (změna implementace je rozbije)
        - „path“ testing
        - mají větší potenciál odhalit mnohem víc chyb
    - **Pozitivní vs. negativní testy**
      - vždy testovat, že:
        - pozitivní – funguje, co fungovat má
        - negativní – nefunguje, co fungovat nemá
          - neomezovat se na „přípustné“ hodnoty, operace, ...
          - vždy zkoušet, jak se SW chová v případě „nepřípustných“ hodnot, operací, ...
          - např. nefunguje přihlášení bez hesla nebo libovolným heslem
- základní techniky
  - techniky testování

- testovací techniky / paradigmatata
  - definují typy testů, které jsou relevantní a zajímavé
  - vytvářejí určitý způsob myšlení a přístup k testování
  - implicitně určují limity, co je relevantní, zajímavé nebo možné
  - existuje velké množství technik, cca 150
  - překrývají se
- jak je využíváme ke tvorbě testů?
  - analýza situace
  - modelování testovacího prostoru
  - volba pokrytí
  - konfigurace testovacího systému
  - provoz testovacího systému
  - pozorování testovacího systému
  - zhodnocení výsledků testu
  - **testovací technika je recept pro provádění těchto činností s cílem objevit něco, co stojí za reporting**
- jak vybrat vhodnou techniku?
  - záleží na několika aspektech
    - požadavky na testy, důvody testování
    - atributy testů
    - kontext vývoje
      - elementy produktu
      - kritéria kvality
      - rizika
      - omezení projektu
  - požadavky na testy
    - najít důležité bugy, aby byly odstraněny
    - pomoci udělat ship / no-ship rozhodnutí
    - ověřit interoperabilitu s jiným produktem
    - minimalizovat náklady na technickou podporu
    - ověřit shodu se specifikací
    - změřit kvalitu
  - atributy testů
    - Power – vysoká pravděpodobnost nalezení problému, pokud existuje
    - Valid – odhalí skutečné chyby
    - Value – odhalí chyby důležité pro uživatele
    - Credible – odpovídá očekávanému chování uživatele
    - Representative – odpovídá tomu, čeho si uživatel nejpravděpodobněji všimne
    - Non-redundant – reprezentuje větší skupinu testů, které se zaměřují na stejné riziko
    - Motivating – „klient“ bude chtít chyby nalezené testem opravit
    - Performable – proveditelný v souladu s návrhem
    - Maintainable – udržovatelný při změnách systému
    - Repeatable – snadno a levně znovupoužitelný
    - Pop (Karl Popper) – odhalí věci týkající se našich základních či kritických předpokladů
    - Coverage – vyzkouší systém způsobem, kterým to nečiní jiné testy
    - Easy to evaluate – snadné a jasné vyhodnocení
    - Supports troubleshooting – poskytuje užitečné informace pro vývojáře, kteří ladí nalezené problémy
    - Appropriately complex – dostatečná komplexnost
    - Accountable – obhajitelnost, prokazatelnost testu
    - Cost – přímé náklady, čas a pracnost
    - Opportunity cost – náklady, které se ušetří provedením testu
- dominantní „techniky“
  - **Function tests**
    - test každé funkce / feature v izolaci

- použití „opatrných“ (middle-of-the-road) hodnot
- neočekáváme selhání
- později „přitvrdíme“
- highly *credible* a *Easy to evaluate* testy
- opomíjí interakce a průzkum přínosů programu
- **Specification-based tests**
  - test SW proti každému tvrzení v referenční dokumentaci (specifikace, uživatelský manuál, ...)
  - závisí na kvalitě referenční dokumentace
    - nutná revize
    - neúplnost, nejednoznačnost, netestovatelnost
  - **Traceability matrix** (pokrytí „tvrzení“ testy)
  - highly *significant* (*motivating*) testy
  - hledá špatně specifikované oblasti
- **Domain tests**
  - testy proměnných (vstupy, konfigurační hodnoty, ...)
  - proměnné v izolaci i ve skupinách
  - všechny přípustné i nepřípustné hodnoty
  - partitioning, třídy ekvivalence, „nejlepší zástupce“
  - pozor na chyby mimo hraniční případy
  - high *power*, Lot of *Information value* testy
  - nižší *Credible* a *Motivating* (corner cases)
- **Risk-based tests**
  - představit si možné způsoby, jak může program selhat, a navrhnout jeden či více testů, které ověří, zda skutečně daným způsobem selže
  - snaha najít „velké problémy“ co nejdřív
  - optimalizace priorit dle míry rizika
  - „kompletní“ množina risk-based testů založena na úplném seznamu rizik (věcí, které mohou jít špatně)
  - nebezpečí špatné identifikace rizik
  - highly *credible*, highly *motivating* testy
  - potenciál mít high *information value*
  - kde hledat problémy?
    - nesplnění některých atributů kvality
    - nové funkce, technologie
    - věci dělané na poslední chvíli
    - unavení, problémoví programátoři
    - nejasností (ve specifikaci, ...)
    - komplexní funkce
    - historicky chybové funkce
    - hlášené problémy ostatních
      - <http://www.bugnet.com>
      - <http://www.cnet.com>
      - odkazy na <http://www.winfiles.com>
      - některé mailing listy a další zdroje ...
- **Scenario tests**
  - testy jako komplexní „příběhy“, které odpovídají způsobu
  - použití programu v reálných situacích
  - nutno dodržet následující atributy:
    - *Complex* (mnoho funkcí)
    - *Credible*, *Motivating* (odpovídá reálnému použití)
    - *Easy to evaluate* (žádná nejasnost, zda test proběhl či selhal)
  - high *power* testy
  - jedná chybná funkce zhatí vše
  - nebezpečí nedostatečného pokrytí

- inspirace – dle produktů konkurence, způsoby chování zákazníka
- varianta „harsher“ testy (killer soap opera ☹)
- **Stress tests**
  - je jasné, že systém při velké zátěži spadne, otázka je, jak, proč a co se stane
  - několik definic
    - zatížit program extrémní aktivitou a sledovat, zda selže
    - testování za hranicemi specifikovaných požadavků na komponentu či program s cílem způsobit selhání systému
    - zahrnutí aplikace nestandardními podmínkami do stavu selhání s cílem sledovat, jak program selže a jaká slabost se tímto odhalí
  - high *power* testy
  - někdy ne tolik *Credible* a *Motivating* testy (netypické pro uživatele)
  - problémy s diagnostikou v případě selhání
- **User tests**
  - testování, které dělají skuteční uživatelé
    - ne testeři předstírající uživatele
    - ne někdo předstírající, že je tester, předstírající uživatele
  - navrhovat testy mohou testeři
  - libovolný test lze použít jako User test
  - důležité je nechat uživateli dostatek „prostoru“
  - *Credible* a *Motivating* testy
- **High volume automated tests**
  - **Random / Stochastic testing**
  - strukturu testu navrhuje člověk, jednotlivé testovací případy vyvíjí, spouští a interpretuje stroj, který následně upozorňuje na selhání
  - nutná maximální (kompletní) míra automatizace
  - jednotlivé testy někdy slabé, málo *Credible* a *Motivating*, není to „řemeslná práce“
  - nutno zabudovat troubleshooting
- **Exploratory tests**
  - Libovolné testování takové, že tester **aktivně** kontroluje návrh testu **během** jejich **provádění** a využívá informace nabyté během testování k návrhu nových a lepších testů
  - Očekávané výsledky, ani konkrétní podoba testů není definovaná, záleží na uvážení testera
  - Používání mnoha stylů, podle toho, který je zrovna nejvhodnější pro splnění cíle
  - Nutný velmi **zkušený tester** se znalostí produktu, domény, testovacích technik, ...
  - Vhodné pro **párové testování**
- **Regression tests**
  - regresní testování není technika sama o sobě, jde o využití testů vytvořených dle jiných technik, zde explicitně vytaženo pro svou důležitost
  - vytváření testů s důrazem na jejich pravidelné opakování po provedení změn v programu
  - libovolný test lze použít jako regresní
  - zpočátku *Power* a *Credible* testy, jejich síla klesá s časem a počtem jejich opakování (pokud se nedějí v systému velké změny)
  - je nutné se naučit navrhovat testovací případy s důrazem na znovupoužitelnost
- plánování testů
  - jak neuspět?
    - zapomenout, že testují lidé
    - předstírat, že testeři jsou odpovědní za kvalitu, nikoliv management
    - diktovat datum spuštění bez ohledu na reálná omezení projektu
    - hodnotit testery podle počtu neulezených chyb
    - nedostatek vzdělávání pro testery
    - oddělit vývoj a testování
  - **plán testů**
    - kolik času vyhradit na testování?
    - co všechno má být v plánu testů?

- **Test coverage** (co testovat)
- **Test methods and tools** (jak testovat)
- **Test responsibilities** (odpovědnosti)
- návrh testovacích případů
  - jak neuspět?
    - malá diverzita použitých technik
      - pouze specification-based testing
      - pouze function testing
    - příliš detailní testovací skripty
      - malá volnost pro kreativitu testera
      - malý prostor pro „náhodu“
      - obtížná udržitelnost
    - exploratory testing bez příslušného vzdělávání
    - oddělení návrhu a provádění testů
    - ignorování existujících rizik
  - jak má vypadat testovací případ?
    - viz techniky testování
    - viz atributy testu
    - míra detailu
    - testovací data
    - standardní struktura
- provádění a vyhodnocování testů
  - provádění testů
    - kdy začít testovat?
      - plánovaný harmonogram vs. realita
        - zpoždění dodavatele
        - čekat nebo začít dříve?
      - dobré časování je zásadní
        - příliš pozdě → problém se splněním termínů, málo času na testování
        - příliš brzo → nestabilní SW, zbytečně vynaložený čas a práce testerů
  - **Trouble ticketing, Bug reporting**
    - základní pravidla
      - evidence **všech** nalezených **issues**
      - jediné místo pravdy
      - nejde jen o to nahlásit issue, důležité je udělat to tak, aby bylo možné jej nasimulovat a **opravit**
      - schopnost odlišit chyby, které „proklouznou“
      - trouble ticketing → bug tracking?
  - reportování výsledků testů
    - standardizovaný test report
      - celkové zhodnocení testovaného SW
      - dopad nedostatků na projekt, systém, ...
      - detailní výsledky – nalezené problémy, odchylky od testovacích případů
      - log testů (provedené testy, průběh testů, ...)
- řízení průběhu testů
  - alokace zdrojů
  - dynamické přidělování a plánování práce
  - reakce na problémy
  - zlepšování procesu testování
  - snaha optimalizovat
  - musíme vědět, „jak na tom jsme“!
    - kdo to ví?
    - jak byste odpověděli na otázku „Kolik testování jste na projektu už udělali?“
    - jak a podle čeho měřit rozsah testování?
    - co je to rozsah testování?

- co je to **rozsah testování**?
  - typicky odpovědi založené na:
    - **produkt**: „Otestovali jsme 70 % řádek kódu.“
    - **plán**: „Provedli jsme 65 % testovacích případů.“
    - **výsledky**: „Našli jsme 753 chyb.“
    - **pracnost**: „Pracovali jsme na tom po 3 měsíce, 60 hodin týdně, provedli jsme 8956 testů.“
    - **kvalita testování**: „Beta testeré našli 28 chyb, které nám unikly, naše regresní testy se zdají neefektivní.“
    - **rizika**: „Dostáváme spoustu stížností od beta testerů, stále máme otevřených přes 500 problémů, produkt nebude do 3 dnů připraven ke spuštění.“
    - **projektová historie**: „V tento moment jsme na předchozích projektech měli 12 % nalezených problémů stále otevřených, stejně by to mělo být i teď.“
  - měření rozsahu testování
    - žádná metrika dokonalá
    - řešením z praxe je kombinace více různých metrik – pokrytí, pracnost, výsledky, rizika, potíže, ...
- automatizace v kontextu testování
  - automatizace exekuce testů
    - snaha automatizovat testy již mnoho desítek let – proč?
      - opakovatelnost a konzistence testů → stejné vstupy a podmínky nezávisle na počtu opakování, odpadá problém s motivací lidí k opakování stejných testů
      - praktická znovupoužitelnost testů → lze opakovat stejný test v různých prostředích, v různých konfiguracích, s mírně modifikovanými vstupními daty a znovuspuštění testů je levné
      - praktické baseline testy → automatizace umožňuje spustit velmi „hutnou“ sadu testů, umožňují efektivně provádět regresní testování
  - automatizace regresních testů
    - velice častý scénář
    - typický průběh automatizace
      - vytvořit testovací případ
      - manuálně jej spustit a ověřit výstup
      - v případě selhání nahlásit chybu
      - v případě úspěchu „uložit“ výsledek
      - opakovaně spouštět test a výsledky porovnávat s uloženými, hlásit chybové situace
      - udržovat automatický test
    - je to skutečně automatizace? co z toho vlastně dělá stroj?
      - analýza programu
      - design testu
      - první spuštění testu
      - uložení výsledků
      - dokumentace testu
      - znovuspuštění testu
      - ← (stroj)
      - vyhodnocení výsledků
      - údržba testu
    - ne pro automatizaci:
      - tvorba testovacích případů je drahá
      - vyžaduje velmi technicky zkušené členy týmu
      - vyžaduje dobře definované a stabilní rozhraní
      - vyplácí se pozdě, výhody automatizace v release N se vrací až v release N+1
      - regresní testy mají často menší Power než nové testy
  - automatizace v kontextu testování
    - kdy může mít smysl? vždy otázka ROI (Return of Investment)!
      - Smoke testing (**Continuous Integration**)
      - Configuration testing (HW SW compatibility)
      - **Variace** (viz. debata o Stochastic testing)

- Stress testing
- Load testing
- **Příprava testovacího prostředí** (data, ...), ...
- nástroje pro automatizaci testů
  - unit a integrační testy: JUnit, TestNG, jMock, EasyMock, DbUnit, ...
  - statická analýza kódu: Findbugs, PMD, JDepend, FoxCop, ...
  - funkční testy – tlustý klient: Selenium, HP QuickTest, IBM Functional Tester, ...
  - funkční testy – tenký klient: HP QuickTest, IBM Functional Tester, White, AutoIt, ...
  - výkonové testy: JMeter, Dieseltest, QALoad, ...
  - komplexní řešení: HP Test Suite, IBM/Rational Test Suite, ...
  - příprava testovacího prostředí: IBM Optim, Grid-Tools DataMaker, Oracle Datamasking, ...
- goodies
  - články
    - HardSwTesting.pdf – **What Is Software Testing? And Why Is It So Hard?**
    - LittleBookOfTesting\_VolumeI.pdf – **Little Book of Testing - Volume I**
    - LittleBookOfTesting\_VolumeII.pdf – **Little Book of Testing - Volume II**
    - SPAWAR\_SwTestPlanGuidebook.doc – **Software Test Planning and Management Guide**
  - checklisty
    - CxCheck\_TestPlan.pdf
  - šablony
    - CxTemp\_TestPlan.doc
    - IEEE\_TestPlanTemplate.pdf
    - MIL-STD-498\_SwTestDescriptionTemplate.doc
    - MIL-STD-498\_SwTestPlanTemplate.doc
    - MIL-STD-498\_SwTestReportTemplate.doc
    - SPAWAR\_SwTestPlanTemplate.doc
    - SPAWAR\_SwTestReportTemplate.doc

## 6. Documentation, Quality Assurance

### Dokumentace

- proč dokumentace?
  - člověk zapomíná, dokumenty nikoliv
  - komunikační prostředek – v rámci týmu, s vedením projektu, se zákazníkem
  - záznamy o domluvách a dohodách – co, kdy, s kým se dojednalo
  - zdroj informací
    - pro uživatele a administrátory systému
    - pro poučení, inspiraci do budoucna (plány, odhady, ...)
    - pro budoucí údržbu
- stručný přehled tématu
  - standardní klasifikace
    - dokumenty procesu
      - popis procesu vývoje a údržby sw produktu – plány, harmonogramy (časové plány), dokumenty týkající se jakosti procesu, projektové standardy, atd.
    - dokumenty produktu
      - popis sw produktu – uživatelská a systémová dokumentace
  - **dokumentace procesu**
    - **plány a odhady** (schedules)
      - používány k odhadům a řízení sw procesu
    - **zprávy** (reports)
      - přehled o používání prostředků během procesu vývoje – dosažena pracnost, plnění termínů, ...
    - **pracovní dokumenty** (working papers)

- často hlavní komunikační prostředek mezi členy vývojového týmu a vedením projektu
- stává se neaktuální → máme problém, co s ní po ukončení vývoje?
  - plány, odhady, zprávy – zdroj know-how pro příští plánování, srovnání s realitou, nezbytné uchovat
  - pracovní dokumenty – často zbytečné uchovávat, nutné dobře zvážit
- **dokumentace produktu**
  - **uživatelská**
    - komu má sloužit?
      - dělení dle zkušenosti
        - zkušení uživatelé – seznam funkcí a jejich popis
        - nezkušení uživatelé – screenshoty, detailní postupy, typické případy užití (scénáře)
      - dělení dle rolí
        - dokumentace pro koncové uživatele
        - dokumentace pro administrátory
    - části
      - Introductory manual
      - Functional description
      - System reference manual
      - System installation document
      - System administrator's manual
  - **systémová**
    - požadavky na systém a jejich důvod
    - specifikace, architektura, design
    - výpis zdrojového kódu s komentáři
    - validační dokumenty (testování, ...)
    - dokumenty údržby (známé chyby, závislost na infrastruktuře, dopady na návrh systému, ...)
- poznatky z praxe
  - správa dokumentace zabere typicky 5 % pracovního času
  - konzistence dokumentace
    - dokumentace je jedna z reprezentací systému
    - dokumentace nabízí mnoho pohledů na systém
    - tyto pohledy se v mnohém „překrývají“
    - → je nutné udržovat všechny reprezentace systému konzistentní
  - forma dokumentace
    - dokumenty – MS Word, MS Excel, PDF, čistý text, webové stránky, wiki
    - diagramy – UML, ...
    - e-maily, záznamy v issue tracking systému (Bugzilla, ...), databázové tabulky, cokoliv užitečného
  - struktura dokumentace – závislá na obsahu, ale existují určitá pravidla:
    - identifikace projektu
    - identifikace dokumentu
    - autor, schvalovatel
    - typ dokumentu
    - aktuální verze, historie
    - distribuční seznam
    - stupeň důvěrnosti
    - abstrakt, klíčová slova
    - copyright
    - seznam pojmů, zkratk
    - obsah, členění na kapitoly, podkapitoly
    - rejstřík
  - orientace v dokumentaci
    - velmi rychlý růst objemu dokumentace
    - primárně dokumenty procesu
    - nutná efektivní správa
    - použití jednotných šablon

- jednotný jazyk
- logická struktura
- jasně určené místo uložení dokumentů
- použití automatických generátorů
- rozcestník – rychlá orientace především pro nové členy týmu
- goodies – templates, checklists
- doporučená literatura
  - NASA\_SwDocumentationStandard.pdf – **NASA Software Documentation Standard**

## Quality Assurance

- stručný přehled tématu
  - **kvalita**
    - „The totality of features and characteristics of a product or service that bear on its ability to meet stated or implied needs.“ (ISO 8402-1986)
      - souhrn vlastností nebo charakteristik produktu či služby, které souvisí s jeho či její schopností splnit explicitně uvedené či implicitně předpokládané potřeby
    - znamená to: **mít spokojeného zákazníka**
  - **quality assurance** – zajištění kvality
    - množina aktivit, jejichž cílem je zajistit kvalitu produktu či služby systematickým a věrohodným způsobem
    - „Quality is never an accident; it is always the result of intelligent effort.“ (John Ruskin)
    - QA nedokáže 100% zajistit tvorbu kvalitních produktů, výrazně však dokáže zvýšit pravděpodobnost, že se tak stane
  - **verifikace, validace** – měření (zjišťování) kvality
    - proces/množina aktivit s cílem zjistit, zda určitý artefakt splňuje nároky na něj kladené
    - validace – building the right thing – kontrola, jestli specifikace určuje, co zákazník chce
    - verifikace – building it right – kontrola, jestli jsme udělali, co je naspecifikováno
  - **testování**
    - proces/množina aktivit s cílem změřit kvalitu vytvářeného software
    - statické testování – review, walkthroughs, inspections, ...
      - **přezkoumání** (review) – projektu, nabídky, designu, kódu
        - velmi efektivní nástroj
        - reviewer kouká z perspektivy mimo projekt
        - šéf dostane zpětnou vazbu z tohoto pohledu
        - odhalení rizik v nabídce
        - oficiální přezkoumání kódu konkrétního člověka – jednou za čas
          - → feedback pro něj pro další profesionální rozvoj osobnosti
        - neoficiální přezkoumání kódu (20% času) – často a pravidelně
          - udělá se jednotka práce – kolega to zkoukne, otestuje, zkritizuje, dá zpětnou vazbu
          - mělo by se s tím počítat v odhadech pracovní
          - mělo by se dělat ihned po dokončení kusu kódu
      - dynamické testování – testování systému za běhu
- proč se zabývat kvalitou?
  - Quality matters!
  - kvalita je finančně efektivní
    - kvalita → nižší cena za údržbu
    - základní cena (za práci samotnou)
    - cena za nízkou kvalitu – náklady na prevenci, na posouzení / zhodnocení, na opravu chyb nalezených zákazníkem nebo při posouzení / zhodnocení
    - často více než 50% nákladů za nízkou kvalitu!
    - náklady – změna během času po zavedení QA
      - chyby → méně chyb (velké ušetření)
      - zhodnocení → lépe zaměřené inspekce a testy (mírně menší náklady)

- prevence → více prevence (mírně větší náklady)
    - náklady na chyby objevené dříve jsou výrazně nižší
  - základní procesy → lepší základní procesy (ušetření)
- kvalita dokáže udržet zákazníky a navýšit zisk
- kvalita je konkurenční výhoda
- kvalita je důležitá pro přežití
- kvalita je důležitá pro mezinárodní marketing
- kvalita je známka „světové třídy“
- Deming's chain reaction: Improve quality → Improve productivity → Decrease costs → Decrease prices → Increase market → Growth in business → Return on investment
- jak na kvalitu?
  - zahájit program kvality → naplánovat program kvality → (implementovat „kulturní“ program, implementovat „technický“ program) → přezkoumat a vyhodnotit → (zase zpět od začátku)
    - zahájit program kvality
      - politika kvality
      - útvar pro kvalitu
      - jasný signál vedení, že: kvalitu podporuje, kvalitu vyžaduje
    - přezkoumat a vyhodnotit
      - revize projektů
      - audity kvality
      - zpětná vazba
      - revize systému kvality
    - naplánovat a implementovat program kvality → quality standard, framework (CMMI, ISO, 6sigma, ...)
      - jak se ohodnotit?
      - co je „dobře“?
      - co je „kvalitně“?
      - nač se zaměřit?
      - jak postupovat?
      - co vyžadovat?
- kvalita a softwarový proces
  - kvalita se prolíná celým procesem
  - V-model
    - na každé úrovni se dělá review plánu a test výstupu
    - při tvorbě plánu se tvoří kritéria pro odpovídající testy
    - Business case → Product verification
    - Requirements → User acceptance tests
    - Functional design → System, integration tests
    - Design & Coding → Unit tests, code analysis
  - SWEBOK
    - Software Quality Fundamentals
      - Software Engineering Culture and Ethics
      - Value and Costs of Quality
      - Models and Quality Characteristics
      - Quality Improvement
    - Software Quality Management Processes
      - Software Quality Assurance
      - Verification and Validation
      - Reviews and Audits
    - Practical Considerations
      - Application Quality Requirements
      - Defect Characterization
      - Software Quality Management Techniques
      - Software Quality Measurement
- poznatky z praxe

- mít globální pohled na softwarový proces v organizaci (stačí přehled na jednu A4)
- QA je nutné naplánovat
- proces musí být pragmatický
  - je dobré dělat kvalitu od konkrétních věcí (odspoda), nedělat vzdušné zámky
    - ale shora musí přijít prohlášení, že vedení kvalitu podporuje
  - o kvalitě je nutné uvažovat na všech úrovních od organizace až po jedince
  - přezkoumání je efektivní (a mnohdy jediný) způsob zajištění kvality
- goodies – templates, checklists
  - SPAWAR\_SQAPlanTemplate.doc – **Software Quality Assurance Plan Template**
- doporučená literatura
  - ESA\_SwEngineeringStandards.txt – **ESA Software Engineering Standards**
  - NAHSyndrome.pdf – **Overcoming the NAH Syndrome for Inspection Deployment**
  - NASA\_FormalInspectionsGuidebook.txt – **NASA Software Formal Inspections Guidebook**
  - NASA\_SwAssuranceGuidebook.txt – **NASA Software Assurance Guidebook**
  - SPAWAR\_SwQualityAssuranceProcess.doc – **Software Quality Assurance Process**

## 7. Configuration management

- stručný přehled tématu
  - podpurná činnost – jedna z nejdůležitějších
  - diagram softwarového procesu (fáze) – řádky Management, Environment
  - definice
    - **softwarový produkt** – úplný soubor počítačových programů, postupů, související dokumentace a údajů (dat), určený pro dodání uživateli
    - **softwarová položka** – jakákoliv identifikovatelná část softwarového produktu v průběžném nebo konečném stadiu vývoje
      - př. třída kódu, SQL skript, dokument, testovací data
    - **konfigurační řízení** – zajištění plného řízení konfigurace softwarového produktu a související dokumentace v průběhu životního cyklu
    - **změnové řízení**
      - součást konfiguračního řízení viděná mechanicky vzhledem k manipulaci se softwarovým produktem
      - jiný název pro řízení rozsahu nad rámec původně domluveného rozsahu
      - jiný název pro model vývoje v době údržby, což je malý Waterfall pro každý change request
      - dva typy změn:
        - chyba – platí dodavatel
        - přání zákazníka (odchylka od dohodnutého) – platí zákazník
- **cíl: zajistit řád a pořádek v konfiguraci softwarového produktu!**
- zajištění cíle
  - zajistit **evidenci** všech částí softwarového produktu – z čeho se produkt skládá?
  - zajistit **identifikaci** všech částí softwarového produktu i celku jako takového
    - mapování verzí položek na verze produktu
  - zajistit, že **provádění změn** softwarového produktu samotný **produkt nepoškodí**
  - zajistit možnosti získat **přehled o stavu** konfigurace softwarového produktu
- pohled ISO 9000-3:1991
  - evidence a identifikace každé softwarové položky
  - evidence a identifikace softwarového produktu
  - schopnost poskytnout údaje pro generování a aktualizaci jednotlivých verzí softwarového produktu
  - schopnost zajistit korektní provádění změn vzhledem k softwarovému produktu
  - schopnost podat zprávu o stavu konfigurace
  - zajištění řízení současné aktualizace dané softwarové položky více než jednou osobou
- pohled SWEBOK – Software Configuration Management (důležité části: Identification, Control, Release)
  - Management of the SCM Process
  - Software Configuration Identification
  - Software Configuration Control

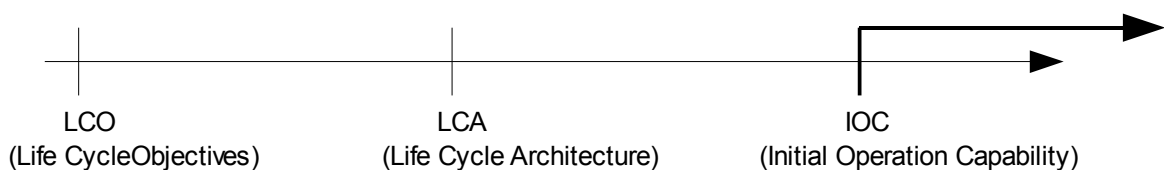
- Software Configuration Status Accounting
- Software Configuration Auditing
- Software Release Management and Delivery
- kontrola verzí
  - identifikace a evidence softwarových položek
    - typy softwarových položek?
    - které spadají pod kontrolu verzí a které ne? (co patří do SVN a co ne?)
      - zdrojáky, XML, menší testovací data → SVN
      - dokumentace → SVN, document management system
      - obří testovací data → sdílený disk (často není vůbec třeba verzovat)
      - cizí manuály, cizí zdroje ke knihovnám → ne do SVN, sdílený disk, zálohovat
    - co s těmi, které pod kontrolu verzí nespádají?
      - obecná data → na sdílený disk a také zálohovat
      - dokumenty → do document management systému (wiki, něco speciálního)
  - identifikace softwarového produktu
    - návrat ke konkrétní verzi – kvůli opravám chyb
    - práce na více verzích současně
  - technická realizace
    - nástroje SCM (Source Code Management), VCS (Version Control System): CVS, SVN, Git, ...
    - prostředky: revision number, tag, branch, ...
  - centralizace vs. distribuce SCM
    - centralizace
      - centralizované repository, lokální kopie klientů
      - každý synchronizuje a vkládá do hlavní větve
      - důraz na **synchronizaci, tracking, zálohování**
      - jediný způsob jak zkusit něco nového je branch → složité
    - distribuce
      - každý svoje vlastní repository
      - vzájemné sdílení na základě „web of trust“
      - centrální hlavní větev? jako centrální uzel explicitně slouží jeden z programátorů, který akceptuje změny od ostatních
      - stažení změny a její aplikace jsou dvě různé operace
      - důraz kladen na sdílení změn
- řízení změn
  - identifikace
    - typy změn
    - vazba na specifikaci
    - vazba na kontrolu verzí
  - proces řízení změn – TODO – ilustrace
  - technická realizace – nástroje: Bugzilla, JIRA, ...
    - priorita – říká zákazník
    - závažnost – říká dodavatel (jak je to velký průser)
- poznatky z praxe
  - jasný a srozumitelný SCM proces
  - verzování softwarových položek i produktu
  - CM a různá prostředí
  - CM a různé typy softwarových položek
  - souběžné verze softwarového produktu
  - CM a různé fáze SDLC
  - správné využití nástrojů
- goodies – templates, checklists
  - NASA\_SwCMPlanTemplate.doc – **NASA Configuration Management Plan Template**
- doporučená literatura
  - ESA\_SwCMGuidebook.pdf – **Guide to software configuration management** – trochu delší

- LittleBookOfCM.pdf – **Little Book of Configuration Management**
- NASA\_SwCMGuidebook.pdf – **NASA Software Configuration Management Guidebook** – krátká a stručná příručka

## 8. Development environment, build process

- stručný přehled tématu
  - motivace
    - váš systém je určen pro cílové prostředí u zákazníka, ale:
      - musíte ho někde vyvíjet a testovat
      - zákazník ho musí někde před nasazením do produkce akceptačně otestovat
      - zákazník může mít více produkčních prostředí
    - **je nezbytná existence dalších prostředí mimo cílové!**
  - typy prostředí
    - u dodavatele
      - **vývojové** – lokální vývoj, povinná sada testů (krátká, rychlá) před commitem do SCM
      - **integrační** – continuous integration (smoke, test, daily build) – rozsáhlejší automatické testy
      - **testovací** – výkonové a jiné nefunkční testy, manuální testy
      - **předakceptační** – regresní testy (automatické a manuální), kvalifikační testy
        - hodí se pro opravy chyb po releasu
    - u zákazníka – **akceptační, produkční #1, produkční #2 (záložní produkční prostředí), ...**
    - příklad možných konfigurací při vývoji:
      - vše je u programátora (aplikační server, databáze, ...)
      - u programátora pouze aplikační server, databáze a IS sdílené
  - kontext projektu – ilustrace
  - dílčí úkoly
    - všechny typy konfiguračních jednotek
    - **zálohy** a jejich obnova
      - přestanou-li existovat sdílené disky, repository, PC lidí na projektu, vývojové prostředí, pak:
        - **ze záloh je možno postavit vývojové prostředí, a to ekonomicky**
      - vyzkoušet obnovu podle plánu
      - prvním krokem je evidence součástí projektu
      - zálohovat zdrojáky i binárky použitých knihoven
        - časem mohou být veřejně nedostupné
        - zdrojáků může být potřeba při debugování (když jsou chyby přímo v knihovnách)
    - evidence a zálohy softwarových položek
    - **dodávky** (malé, velké)
    - postupy (instalace, modifikace, přístupy ...)
    - záloha použitých zdrojových kódů
    - příprava dokumentace pro nové členy týmu, aby se rychle mohli zorientovat v projektu
    - audit, ...
  - co také znamená „prostředí“
    - příjemné pracovní prostředí – nábytek, soukromí, klid, dostatečný prostor, ... (kanceláře vs. open space)
    - kvalitní nástroje – PC nebo notebook, IDE, knihovny, ...
  - dodávka
    - souvisí s CM – dodávka vytváří branch
    - co musím umět:
      - vyrobit dodávku
      - nainstalovat dodávku
      - připravit dodávku pro instalaci zákazníkem
      - dodat systém jako celek
      - opravit malou drobnost – a opravit ji rychle a ekonomicky
      - poradit si s různými typy prostředí – aplikační server, databázový a replikační server, OS, ...
    - podstatné pojmy

- release (build)
- oprava buildu (patch)
- dodávka – malá (drobná oprava), velká (pravidelná aktualizace)
- instalační set
  - en-bloc – lze nainstalovat na zelené louce (např. aplikace)
  - inkrementální – závisí na předchozí verzi (např. databázový skript)
    - př. „nakopíruj tuhle classu na toto místo“
    - databázové skripty musí být znovuspustitelné (nesmí záviset moc na předchozím stavu)
      - př. *pokud sloupec neexistuje*, vytvoř ho
    - zákaznickova data jsou to nejdůležitější, co má
  - dodávka se může skládat z více instalačních setů (aplikační části, databáze, nastavování)
- poznatky z praxe
  - maximálně věrné prostředí vývojové, testovací, akceptační, ...
  - denní build (continuous integration)
    - je dobré automaticky buildovat každý den nebo několikrát denně
    - je důležité sledovat výsledky testů: selhání → nutná oprava
  - jednoduchý a automatizovaný proces dodávek
  - kontrolované zálohovací logy a reporty automatických testů
  - př.: vhodné kroky při updatu databáze:
    - zakázat přístup z proxy
    - vypnout replikace
    - updatovat aplikační server
    - znovu nahodit replikace a proxy
  - popis instalace musí být stručný
- minimální nároky
- ilustrace
  - vývojové prostředí a proces dodávky
    - SVN jako SCM
    - Ant jako build nástroj
    - CruiseControl jako integrační platforma
      - každodenní build, deploy, test
      - každodenní kontrola zdrojových kódů dostupnými nástroji
      - rychlá zpětná vazba v případě problémů
      - neustálá „integrace“ (Continuous integration)
  - když instaluje zákazník ...
- templates, checklists
  - complete\_list\_of\_sched\_risks.txt – **Complete List of Schedule Risks**
- doporučená literatura
  - Steve McConnell: **Daily Build and Smoke Test, Best Practices**, IEEE Software, Vol. 13, No. 4, July 1996
    - <http://www.stevemcconnell.com/ieeesoftware/bp04.htm>
  - **Continuous Integration**, Martin Fowler
    - <http://www.martinfowler.com/articles/continuousIntegration.html>
  - Clark, M. Pragmatic **Project Automation: How to Build, Deploy, and Monitor Java Applications**. The Pragmatic Programmers, 2004.



- stručný přehled tématu
  - co je údržba?
  - stav systému

- systém je dodán v rozsahu dle nabídky
- systém je akceptován a rutinně provozován
- systém neobsahuje příliš mnoho chyb
- předmět vývoje
  - chyby / problémy v produkci
  - drobné změny
  - **systematický rozvoj**
- rytmus dodávek
  - **pravidelné plánované releases**
  - malé dodávky pro naléhavé věci
- od IOC se o software začínáme dělit se zákazníkem a uživateli
- nemalé množství byrokracie
- typy údržby dle ISO/IEC 14764
  - Corrective – za účelem opravy nalezených chyb a problémů
  - Adaptive – za účelem udržení použitelnosti software v měnícím se prostředí
  - Perfective – za účelem zlepšení výkonnosti nebo udržovatelnosti
  - Preventive – za účelem detekce a opravy latentních chyb dříve, než se z nich stanou chyby skutečné
- SWEBOK
  - Software Maintenance Fundamentals
  - Key Issues In Software Maintenance
  - Maintenance Process
  - Techniques for Maintenance
- údržba vs. ...
  - údržba vs. SDLC
    - iterativní modely – používají se většinou při základním vývoji
      - zákazník něco uvidí a uvidí to často
      - řekne, co je špatně, a to co nejdřív
    - pro jednotlivé změny v údržbě resp. změnovém řízení se spíš používá malý waterfall
  - údržba vs. měření
    - velmi snadno lze získat přesná čísla – absolutní, relativní
    - nutné pro dobrou ekonomiku
    - podklad pro servisní smlouvu na další léta
    - čas, úsilí, kvalita (Bugzilla), rozsah (SVN)
  - údržba vs. dokumentace
    - v rámci údržby upravujete systém:
      - potenciálně dlouho poté, co byl vytvořen
      - aniž byste byli jeho autory
      - to je nemožné bez kvalitní dokumentace!
    - potřebujete minimálně:
      - kvalitní specifikaci, abyste uhlídali rozsah
      - architekturu a design, abyste je mohli ctít
      - jasné a přesné postupy, abyste se jimi mohli řídit
    - otázky:
      - forma
      - množství, uspořádání a orientace
        - je třeba neustále se snažit minimalizovat množství dokumentace
        - ale tak, aby v ní bylo vše potřebné
      - ekonomie tvorby, údržby a používání
    - dokumentace produktu i projektu typicky v SVN
      - dokumentace pro každý bug/change request
    - specifikace – původní, změnová
      - není rozumné pro změnové řízení měnit původní dokument
      - nikdo se pak ve změnách nevyzná
      - změnová specifikace umožní vysvětlit důvody změn a dodat další informace

- údržba vs. vývojové prostředí
  - maximálně podobné produkci – do všech detailů
  - maximálně podobně používané
  - Continuous integration, Smoke testing
- údržba vs. architektura
  - rozšiřitelnost, udržovatelnost, ...
  - nové požadavky
  - plíživé ničení architektury
- údržba vs. konfigurační řízení
  - evidence všech požadavků zákazníka
  - definovaný proces změnového řízení
  - mapování na dodávky
  - klasické situace
    - práce na dalším release
    - oprava chyby v akceptaci - nekritické
    - oprava chyby v produkci – kritické
  - obsah dodávek je vhodné archivovat
- údržba vs. testy
  - větší systém je prakticky nemožné po každé změně otestovat ručně celý, takže:
    - komplexní testování se buď ignoruje (špatně) nebo lépe:
      - existují regresní automatické testy
      - průběžně se testuje řešení každé chyby resp. změny
      - testy jsou dobře naplánované a zorganizované
      - existuje záznam o testování
      - existuje dobré akceptační testování
  - testy je možné zadat do Bugzilly jako úkoly
- údržba vs. ekonomika
  - cílem je na údržbě vydělat, musíme mít:
    - velmi efektivní proces
    - velmi přesné odhady
  - je potřeba namapovat evidenci práce na fakturaci dle servisní smlouvy
    - odlišit změnové řízení (platí zákazník) a údržbu (platí dodavatel)
    - různé činnosti: analýza, implementace, project management
  - rozpočet zákazníka na údržbu
    - paušál na servis – telefonní linka, opravy do ...
    - balík na další funkcionalitu
      - zákazník má připraveny svoje požadavky
      - dodavatel nabízí své nápady
    - na to potřebujeme znát data – průměrný koláč nákladů, náklady na analýzu, ... → měření
- údržba vs. vše ostatní
  - v období údržby je kladen zvýšený důraz na kvalitu a efektivitu prakticky všech činností
  - je těžké udržet pořádek
  - je snadné polevit
  - je snadné údržbu podcenit
  - je snadné „šlápnout vedle“
  - příklad s Bugzillou
    - hlavní chyba → více malých chyb pro více lidí
      - problém: Bugzilla neumí agregovat pracnost (bug → dodávka → systém)
      - je třeba další nástroj na evidenci práce
- rekapitulace
  - systém, agendu, situaci, zákazníka – známe
    - lze mít velký pořádek v procesu údržby
    - lze přesně stanovit okrajové podmínky
    - lze přesně určovat pracnost, data dodávek, ...

- systém, agenda, situace – jsou velké / složité
  - lze elegantní systém postupně snadno rozbít
- typické problémy
  - podcenění
  - opomenutí
  - chuť "vydělat" na parciální věci
  - závislost na konkrétních lidech
  - ne každý se údržbou dokáže nadchnout
  - plíživé ničení architektury, designu, ...
  - "čemu nerozumím mažu"
  - neznalost systému
  - a další
- poznatky z praxe
  - disciplína a dodržování rozumných postupů je nutnou podmínkou
  - zásadní závislost na kvalitě návrhu
  - čím složitější systém, tím složitější údržba
  - velmi složité s osobami, které nemají k systému "citový" vztah
  - nepodceňovat moment únavy
- minimální nároky
- templates, checklists – vše z ostatních kapitol
- doporučená literatura
  - sekce týkající se maintenance ze základních monografií:
    - Pressman R. **Software Engineering: A Practitioner's Approach**. 5th ed., McGraw-Hill, 2001
    - Dorfman, M. and Thayer, R. **Software Engineering**. IEEE Computer Society Press, 2000
    - Sommerville, I. **Software Engineering**. 8th ed., Addison Wesley, 2006

## 10. Software project management

- stručný přehled tématu
  - co dělat, když začneme mít pod sebou více lidí?
  - co všechno se může skrývat pod pojmem Software project management?
  - obsah PM
    - PM dle **SWEBOK**
      - Initiation and Scope Definition
      - Software Project Planning
      - Software Project Enactment
      - Review and Evaluation
      - Closure
      - Software Engineering Measurement
    - PM dle **NASA**
      - začátek plánování projektu, porozumění obsahu práce
      - definování technického přístupu
        - výběr a adaptace vhodného modelu životního cyklu
        - výběr vhodných aktivit, metod a produktů
      - dokončení plánu projektu, definování přístupu k vedení
        - organizace, odhadování, časové plánování, ...
      - provádění projektu (vykonání SW plánu projektu)
        - monitorování, řízení (Control), údržba SW plánu, ...
      - uzavření projektu
    - PM dle **Sybase**
      - Initiation – definice problému a možných řešení, naplánování projektu
      - Execution – provádění plánu, monitorování a řízení vývoje (progress)
      - Closeout – ukončení projektu
    - PM dle **best practices** – Program Control (Software Management Program Control) vyžaduje:

- plánování (Software Management Planning)
  - definování cílů na produkt
  - strukturování projektu
  - přezkoumání plánu
  - časové plánování (scheduling) projektu
  - testování plánu
  - ocenění nákladů (costing the plan)
  - časté revidování plánu podle změn okolností projektu
- provádění aktuálního plánu
- zahrnování změn do plánu a projektu
- dosahování cílů týkajících se produktu, zdrojů, kvality
- koordinování úsilí skupin a jednotlivců
- zajišťování adekvátních kontrol kvality k podpoření vysoké kvality softwaru
- diagram softwarového procesu (kulatý) – Project management se dotýká všech částí
- diagram softwarového procesu ( fáze) – řádek Management (PM + konfigurační řízení)
  - první kopeček – zafixování požadavků
  - druhý kopeček – určení počtu lidí do týmu, další zdroje
  - PM je ale hlavně průběžná činnost
- silné vazby PM na: Requirements, Architecture, Configuration Management, Engineering Process, Quality
- PM prakticky - jak na vedení projektu?
  - **základní metoda**
    1. ustavení organizace projektu
    2. zjištění a porozumění „práci, co se má dělat“
    3. postupné rozložení „práce“ na jednotlivé, dostatečně kompaktní jednotky
      - provést odhady
      - přiřadit omezení a nároky na zdroje
      - provést analýzu závislostí, ...
        - existuje WBS (Work Breakdown Structure), harmonogram, activity network, ...
    4. přiřazení zdrojů
    5. měření, monitorování, zaznamenávání a hlášení postupu, přijímání nápravných opatření
    6. zaznamenání a uložení údajů o projektu pro další použití
  - základní metoda v kontextu
    - „základní metoda“, rekurzivní aplikace „základní metody“
    - model životního cyklu vývoje software (SDLC)
    - struktura softwarového systému (architektura)
    - průběžné a postupné plánování
  - PM v kostce
    - musíte vytvořit a udržovat plán s výhledem do dostatečně daleké budoucnosti
    - musíte mít jasno v termínech a závazcích vašich a třetích stran
    - vaši lidé musí vždy přesně vědět, co mají dělat v nejbližších dnech (cca týden), aby si mohli sami zorganizovat práci
    - musíte dbát o efektivitu práce vlastních lidí a mít vše dobře zorganizované, měřit a přemýšlet o ekonomice projektu, poučit se rychle z vlastních a cizích chyb
    - musíte rozumět systému, chápat jeho složitost a umět se rozhodovat, když vás zákazník tlačí na nějaké schůzce k závazku na rozsah, termíny, ...
    - musíte komunikovat se zákazníkem, chodit na schůzky, psát zápisy, předávat protokoly, ...
    - musíte mít přehled a evidenci všech problémů a rizik a názor na jejich řešení resp. eliminaci
    - musíte být stále na pozoru a hlídat rozsah! (scope creep)
    - př. Breathalyzer Test
- vybraná zajímavá témata
  - poznámky z praxe
    - je třeba sednout si a sepsat si myšlenky a rozpracovat je
      - lepší málo než nic
      - např. údržba plánu: jednou za týden schůzka → aktualizace pár řádků, vystavení aktuálního plánu na

web

- project manager
  - je to hlavní autorita, musí rozumět zadání
    - je šéfem projektu, ne jen koordinátorem – má zodpovědnost za projekt
  - iniciuje práce na zadaném projektu
  - musí být schopen rozhodovat
    - musí mít pro to informace a know-how
    - musí mít přehled, aby mohl odhadovat pracnost
    - musí mít selský rozum
  - komunikuje se zákazníkem – je proxy mezi zákazníkem a týmem
- výběr životního cyklu je velmi důležitý
  - je třeba dohodnout se se zákazníkem – v jeho jazyce
  - plánování práce vychází z životního cyklu (např. rozdělení na analýzu a implementaci, iterace)
  - dnes je typické rozdělení do mnoha kratších fází (měsíce, čtvrtletí) – je to efektivnější
    - čím dřív se odhalí problém, tím lépe
  - s rozdělením typicky přichází zákazník
- často dělá analýzu jeden dodavatel a implementuje jiný (business analýza → softwarová analýza + implementace)
  - problém: nalezení chyb v předchozí části – nutnost iterace
- uzavření projektu
  - analýza, zhodnocení metrik, historie
  - co si firma může z projektu vzít?
  - co zbyde z projektu?
    - peníze, údržba, zkušenosti, data
    - ale hlavně odhady pracnosti pro další projekty!
- nástroje – cokoliv použitelného – Excel, Word, MS Project, wiki, ...
- plán
  - kritéria iterací – interní obdoba akceptačních kritérií
  - obsah plánu – úvod, plán konfiguračního řízení, plán dokumentace, plán instalace, ...
  - části plánu se vyvíjejí v průběhu projektu (např. instalace)
  - problém ~ riziko, které se stalo
- WBS (Work Breakdown Structure) – hierarchie činností
  - snažíme se systém rozložit postupně na strom
  - cílem je dostat se na listy a ohodnotit je pracností
  - je důležité využít architekturu
  - vzniká komunikací základních členů týmu (PM, architekt, ...)
  - WBS + omezení → počty a profily lidí do týmu (staffing)
    - ve firmě je typicky pool volných lidí – ale o ně je boj
    - často je nutné si zamluvit lidi dost dopředu
    - lidí je typicky málo, a tak dělají na dvou projektech zároveň
  - WBS a harmonogram jsou zásadní – neustále jsou upřesňovány
- výhled
  - podřízení musí vědět, co mají aktuálně dělat
    - aspoň týden dopředu
    - týdenní plán se musí aktualizovat každý den
  - hrubý plán na několik měsíců dopředu – každý týden úpravy
- activity network – síť závislostí
  - v týmu o 5-7 lidech typicky není moc potřeba, práci lze rozumně paralelizovat
- problémy, rizika – vědět o nich a řešit je
  - např. rizika týkající se rozsahu
    - zákazník si pořád něco vymýšlí a snaží se to do software protlačit přes programátory
  - PM řeší programátorské problémy místo podřízených
- měření / řízení – virtuální „panel“ pro měření a řízení projektu (Project Control Panel) – v hlavě :)
- modely SDLC – Waterfall, iterativní (SCRUM, Spiral Model), evoluční (Recursive SDLC), ...

- minimální nároky na softwarový plán
- templates, checklists
  - Complete List of Schedule Risks
  - Most Common Schedule Risks
  - Project Definition template
  - Risk Management Plan template
- doporučená literatura
  - **Anchoring the Software Process**
  - **SAFE Project Definition Document**
  - **NASA Management Guidebook**
  - **SPMN Best Practices Guidebooks**
  - **The Joel Test: 12 Steps to Better Code**
  - RecursiveSDLC.pdf – **A Proposal for a Recursive Object Oriented Life-Cycle**
  - Brooks, F. P.: **The Mythical Man-Month**
  - DeMarco, T.: **Peopleware**
  - Metzger, P., et al.: **Managing a Programming Project: People and Processes**
  - McConnell, S.: **Software Project Survival Guide**
  - Royce: **Software Project Management: A Unified Framework**

## 11. Proposals, estimation, project history, measurement

### Nabídky, odhadování

- varianty nabídky:
  - nový systém
  - úprava existujícího systému – mimo standardní rozsah změnových řízení, cizí systém
- proces tvorby nabídky
  - evidence nabídky – hodí se i Bugzilla :)
  - odpovědnost
  - tvorba nabídky
  - přezkoumání
  - komunikace
  - evidence pracnosti
- tvorba nabídky
  - nároky na obsah nabídky
  - instrukce ke stanovení rozsahu
  - přístup k odhadování
    - počty (obrazovky, tabulky, ...) / FPA, analogie, cenový model, kombinace všeho
  - problémy: málo času (např. 2 dny)
- obsah nabídky
  - nejlepší a nejpřesnější údaje – rozsah, pracnost, termíny, kvalita, nároky na zdroje, rizika, okrajové podmínky
  - nároky na zdroje – i na konkrétní lidi → je dobré omezit platnost nabídky
    - volných lidí v poolu je nedostatek, projekty se o lidi mohou prát
  - dokument nabídky – někdo čte jen Management Summary
  - plán – rozumný kompromis v termínech (nereálné požadavky → nestíháme vs. naše nabídka je vyloučena)
  - ať se mohou mocní rozhodovat :)
- stanovení rozsahu
  - rozsah je stanoven taxativně a strukturovaně
  - pozitivní i negativní vymezení
    - funkční požadavky – co je v základu aplikace vs. co bude rozšiřující modul
  - vymezení formou počitatelných věcí
  - defenzivní forma
  - metoda „budoucího upřesnění“

- všechny typy požadavků – mít checklist i na to, na co zákazník nepomyslel
- okrajové podmínky
- nároky na postup vývoje, dokumentaci, ...
- požadavky na spolupráci
- odhady
  - historie projektů
  - cenové modely (např. CoCoMo – je třeba upravit pro své podmínky)
  - zkušenosti
  - lidé, kteří dělali podobné věci
  - cokoli dalšího, co pomůže

## Historie projektů

- proč vytvářet?
  - nabídky – odhady, okrajové podmínky, rizika, problémy, ...
  - údržba
  - ekonomika
  - po letech je schopnost odhadovat často z projektu to nejzajímavější a nejcennější
- co je obsahem?
  - celková pracnost, kalendářní čas, počty lidí
  - pracnost dle typů činností
  - kalendářní čas dle typů činností
  - počty (obrazovky, tabulky, tisky, programy, ...)
  - charakteristika systému a agendy
  - problémy, rizika
  - vše, co je vhodné uchovat pro budoucnost
- jak vytvářet?
  - jednoduše, přehledně, schematicky
  - naměřená data ukládat standardně, aby bylo možno porovnávat různé projekty
  - nutnou podmínkou je existence naměřených dat!

## Měření

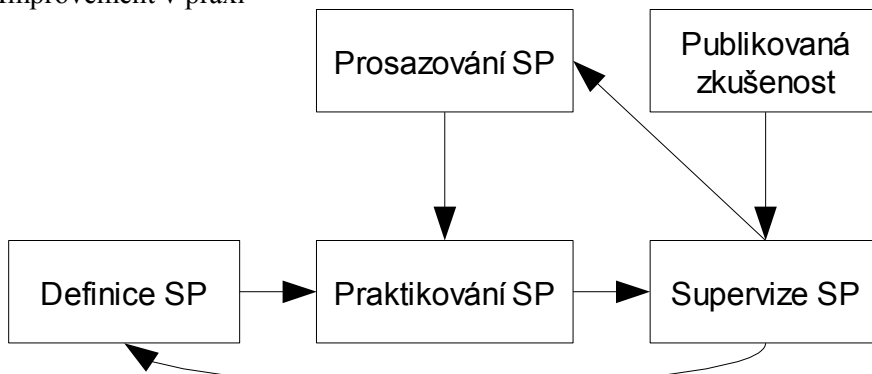
- proč měřit?
  - pro odhad časové pracnosti
  - nutné pro dobrou ekonomiku
  - historie projektů
  - tvorba nabídek – nutné pro argumentaci při vyjednávání cen
  - tvorba servisní smlouvy, ...
- co měřit?
  - základní metriky:
    - time (kalendářní čas) – z evidence práce
    - size (rozsah) – počet řádek kódu, CVSstat
    - effort (pracnost) – z evidence práce
    - quality (jakost) – Bugzilla, Jira
- jak měřit?
  - velmi snadno lze získat přesná čísla: absolutní, relativní
  - lze použít elementární mechanismy
    - Bugzilla, Jira, ...
    - CVSstat, SVNstat, ...
    - libovolný systém pro vykazování času
- doporučená literatura
  - **Evidence based scheduling**, Joel Spolsky
  - Coombs, P. **IT Project Proposals: Writing to Win**. Cambridge University Press, 2005.
  - McConnell. **Software Estimation: Demystifying the Black Art**. Microsoft Press. 2006.

- **Practical Software and Systems Measurement: A Foundation for Objective Project Management**, v. 4.0b
- **COCOMO (II)** (COConstructive COSt MOdel)

## 12. Software process

- základní pojmy
  - Softwarový proces
    - množina aktivit, praktik, metod a transformací, které lidé používají k vývoji a údržbě SW a souvisejících artefaktů (plány, dokumentace, ...).
  - Software Process Improvement
    - postupné zlepšování softwarového procesu ~ QA
  - Software Process Engineering Group (SEPG)
    - skupina lidí, kteří stojí v centru snahy všech jedinců ve firmě zapojených do zkvalitňování SW procesu.
    - skupina obvykle odpovídá velikostí 1 – 3 % velikosti vývojového oddělení.
- základní koncept
  - **PDCA model** (Deming cycle)
    - Plan
      - prověřit současnou výkonnost
      - posoudit problémy, omezení
      - navrhnout řešení
      - naplánovat provedení
      - typicky se jedná o malé věci: zavést administrátorskou dokumentaci, regresní testování, ...
      - důležitý krok: porozumět metrikám → jak na tom vlastně jsme?
    - Do
      - otestovat účinnost řešení
    - Check
      - zhodnotit výsledky testu
      - posoudit dosažení výsledků
      - zaměřit se na překážky bránící zlepšení
      - změřit výsledky, feedback
    - Act
      - rozpracovat konečné řešení aby bylo kdekoli použitelným přístupem
      - co je těžké?
        - prosadit změny do praxe
        - rozhodnout, které metriky a problémy jsou důležitější, co vybrat k řešení
- různé metodiky/modely
  - rozdělení:
    - systematické
      - preskriptivní – někdo nám nařizuje, co je pro nás nejlepší
      - induktivní – sami hledáme pomocí rad inspiraci ve vlastním prostředí (firmě)
    - best practices – checklisty, nejvíce používané v praxi
  - co použijeme, na tom až zas tak moc nezáleží
  - typický scénář: začneme s best practices a časem to třeba namapujeme na ISO (kvůli státní zakázce)
  - ISO
    - ISO 9001:2000 – abstraktní, nejen pro software
    - **ISO/IEC 9003:2004** – aplikace 9001 pro software, stále dost abstraktní
    - vždy je potřeba mapovat na realitu ve firmě
  - **CMM/CMMI** (Capability Maturity Model Integration) – USA
    - levely: Initial, Repeatable, Defined, Managed, Optimizing
    - v ČR mají firmy max. tak level 3, jenom pár má level 4, level 5 má jen pár firem ve světě
    - CMM level 3 ~ ISO 9001:2000
    - výtky k CMM: rozdělení – software vs. lidi (bez integrace)
      - → CMMI = CMM Integration – obecně použitelné
      - úrovně pro různé oblasti

- **NASA SEL** (Software Engineering Laboratory)
  - základní premisa – určitá vývojová organizace musí úsilí o zkvalitňování zaměřovat na zamezení minulých problémů a na opakování minulých úspěchů
  - Understanding ~ Plan z PDCA
  - Assessing – výběr pár projektů, na kterých zkusíme vylepšení
  - není zde standard, co je dobře či špatně → není zde ani certifikace
- **SPMN best practices**
  - přímo použitelné strategie, techniky a praktiky
  - Little book of ...
  - stručné, koncentrované, ideální na nástěnku :)
  - pro praxi je vhodné použít výcuc ze SPMN
- shrnutí – co je třeba?
  - znát současný stav vlastního procesu vývoje a jeho charakteristiky
  - znát problémy s ekonomickým projevem a jejich vážnost
  - mít názor, které problémy je nutné a možné odstranit
  - mít názor, jak modifikovat proces vývoje
  - mít prostředky, jak tuto modifikaci prosadit
  - ... znovu na začátek
- změna
  - proč? co přinese? jak verifikovat výsledek?
  - prostředky – respekt, moc
    - přesvědčit kolegy o dobrých nápadech
    - typicky změny fungují odspoda, pak to akorát posvětit management
- Software Process Improvement v praxi



- **definice**
  - základní koncept
    - zavést nový element procesu pouze když:
      - je identifikován problém,
      - jedná se o vážný problém (určitým způsobem),
      - element procesu byl úspěšně zaveden a prověřen na úrovni projektu.
    - lepší je mít v procesu méně věcí, než ho mít nabobtnalý (stačí jedna A4 na každou oblast)
  - způsob definice
    - minimální praktiky – málo direktivní
      - minimální a kompaktní sada obecných pravd, které jsme identifikovali jako relevantní pro nás
      - původně nejmenší společný jmenovatel všech závažných problémů v naší organizaci
      - checklist, komentáře, poznámky, goodies
    - politiky (policies) - direktivnější
      - snaha neopakovat problematické věci, kterým lze snadno zabránit, stále dokola ...
      - jednoduché textové soubory v direktivní formě
    - standardy
      - netypické, vznikají pouze ve výjimečných případech
      - př.: SQL procedury v Oraclu, Java třídy
- **praktikování** – ve zkratce

- project centered
- tailoring SW procesu pro projekt
  - mapování – jak budou konkrétně vypadat minimální nároky pro daný software – details
- individuální postupy na úrovni projektu
- CVS, SVN, Bugzilla
- historie projektu, hlavní stránka projektu
- sledováno využití zdrojů, základní měření
- sledovány rizika, problémy
- finanční tabulka
- individuální zapojení jedinců – kariérní řád
- pravidelné, strukturované revize projektů
- **supervize**
  - pravidelné revize zákaznického týmu
    - revize znalostí SAMa (SAM = Services Account Manager)
      - SAM – technický ekv. obchodníka, vedoucí zákaznického kontextu
    - revize využívání zdrojů a plánování
    - individuální revize konkrétního projektu
    - identifikace znovupoužitelných věcí, nápadů, ...
    - distribuce informací o ostatních projektech (integrace) přes člověka, co dělá revize
  - pravidelné revize projektů (je-li to nutné) – detekce problémů s minimálními nároky
  - finanční tabulka
  - CVS / SVN
  - Bugzilla (defekty, issues, změnová řízení, ...)
  - interní systém se základními metrikami pro pracnost
- **prosazování**
  - kariérní řád (teorie, praxe, proces)
    - kdo chce postoupit, musí nastudovat potřebné věci a ještě udělat něco prakticky užitečného
    - cíl: aby člověk sám chtel dělat to, co je potřeba
  - školení, odborné zdroje
  - přiřazení lidí k práci
  - plánování zdrojů
  - přezkoumání (např. zdrojového kódu)
  - proces schvalování nabídek
  - intranetové stránky
  - „Úterky“ - pravidelné schůzky lidí z celé organizace
  - prosazování změn do praxe může trvat i třeba 10-15 let
- charakteristiky procesu (příklad v Profinitu)
  - kvantitativní
    - chybovost ... 2 chyby / 1KSLOC
    - produktivita ... 14-17 SLOC / MH (man-hour) – začátkem projektu i 40-50, ale při údržbě méně
  - praktické
    - dodržování termínu je standard
    - kontrahovaná pracnost odpovídá vynaložené
    - zákazníci si nestěžují, resp. explicitně oceňují kvalitu
  - kvalitativní
    - ISO 9000-1
    - nejsou problémy s termíny, kvalitou, vlastnostmi systému.
- náklady – pokud je SPI už zavedeno do praxe, během zavádění mohou být vyšší
  - přímé – SEPG ... 2 %
    - Review ... maximálně 0,5 %
    - Úterky ... maximálně 1,2 %
    - Revize ... maximálně 0,25 %
  - nepřímé (kariérní postupy, Objectives, ...)
  - čím více je to „normální“, tím méně to stojí extra vykazatelného času!