Softvérové inžinierstvo

- disciplína, ktorá sa zaoberá analýzou, špecifikáciou požiadaviek, návrhom, implementáciou, testovaním, nasadením a používaním softvéru
- systematický prístup k vývoju, prevádzke, údržbe a vyradeniu softvéru

Prečo je štúdium softvérového inžinierstva dôležité?

- zvyšovanie produktivity softvérových inžinierov (analytikov, návrhárov,...)
- zlepšovanie vlastností softvéru, najmä spoľahlivosti, bezpečnosti a použiteľnosti
- softvér je všade okolo nás

softvérová kríza + softvérové inžinierstvo

- konferencie NATO v rokoch 1968-1969
- vznik softvérového inžinierstva bol dôsledok softvérovej krízy
- treba vytvoriť softvér podľa **požiadaviek** s ohraničenými **zdrojmi** a určeným **časom** ukončenia

kvalita

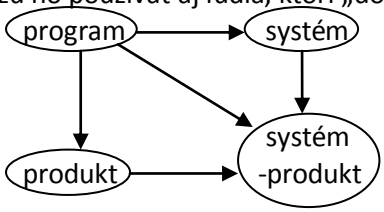
- súhrn vlastností a charakteristík výrobku, procesu alebo služby, ktoré preukazujú jeho schopnosť splniť určené alebo odvodené potreby
- stupeň splnenia požiadaviek, resp. potrieb

pojmy

- metóda - postup
- technika - spôsob
- prostriedok - nástroj
- metodológia - súbor metód a techník

softvér

- zbierka počítačových programov, procedúr, pravidiel a s nimi spojenou dokumentáciou a údajmi
- program = vytvoril 1 človek pre 1 človeka
- systém = niečo zložitejšie, pozostávajúce z častí
- produkt = môžu ho používať aj ľudia, ktorí „do toho nevidia“ (dokumentácia...)



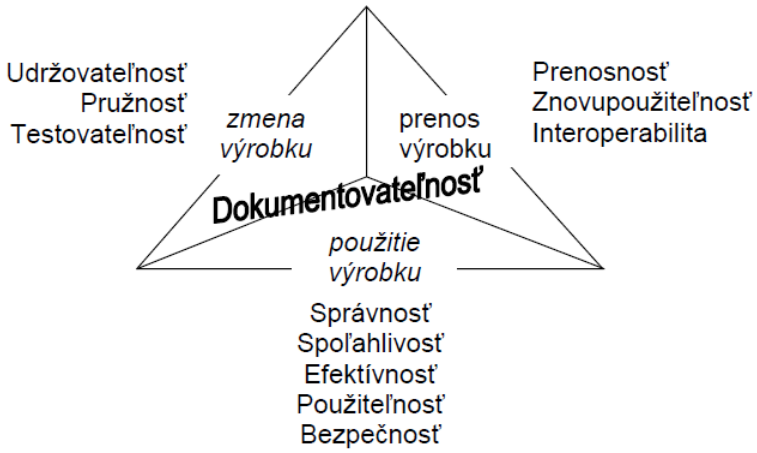
škálovateľnosť

- niečo čo funguje v malom nemusí fungovať aj vo veľkom

typy softvérových výrobkov

- generické – nemá konkrétneho zákazníka
  - „off-the-shelf“ označuje „hotový“ produkt, ktorý už nepotrebuje ďalší vývoj
    - COTS – commercial – obvykle zakúpený finálny produkt, pripravený na použitie
    - MOTS – modified – produkt, ktorý po obstaraní musíme prispôsobovať (modifikovať) a až potom je pripravený na použitie
- zákaznícke – na objednávku

vlastnosti softvéru



problémy s tvorbou softvéru

- podstatné vnútorné problémy
  - zložitosť – riešenie: dekompozícia
  - prispôsobivosť – softvér sa musí prispôsobiť okoliu
    – riešenie: voľná zviazanosť (loose coupling)
    - pripomínam open-closed principle z OOP: softvérové entity majú byť otvorené pre rozširovanie ale uzavreté pre zmeny (voľný preklad: „chcem pridať ďalšiu featurku, tak kódnem do triedy nové funkcie, pri dobrom návrhu nemusím meniť tie staré“)
  - nestálosť
  - neviditeľnosť (nevizualizovateľnosť – v súvislosti s modelovaním)
    - neviem popísať softvér predtým ako vznikne, neviem ani presne, čo mi ešte chýba
    - s tým spojený syndróm „90% hotovo“ – kódil som to 20 hodín a myslím si, že už to mám takmer hotové (na 90%) lenže tých posledných 10% (zlý odhad) mi zaberie ešte ďalších 20 hodín k totálnemu dokončeniu (extrém)
    + syndróm „druhého projektu“ – keď robím niečo 1. krát, tak som dôkladný, dávam si pozor – keď mám spraviť podobnú vec znovu, tak to podceňujem kvôli tomu, že som niečo také už robil -> dopadne to horšie ako 1. projekt
- nie zákonité problémy
  - špecifikácia požiadaviek (nejasnosť, nestálosť, nekonzistentnosť, neznalosť)
  - programátorská produktivita (niekto by to kódol 20x rýchlejšie)
  - slabá opakovateľnosť pri tvorbe (kódim to manuálne všetko ako debil miesto toho, aby som využil nejaký framework a nejaké už existujúce veci...)
  - náchylnosť softvéru na chyby
  - práca v tíme – najmä organizácia práce
  - dokumentácia – najmä jej nedostatok pri údržbe (viď. Softvér na riadenie podnikových zdrojov, ktorý prednášal A.Danko)
  - problém mierky – škálovateľnosť
    - kódol som niečo, čo pri 2 užívateľoch fungovalo dokonalo, keď sa tam pripojilo 50 ľudí, tak to zdochlo
  - neskoré odhalenie chyby – čím skôr (pri vývoji) nastala chyba, tým horšie sa opravuje
  - starnutie softvéru
    - dôležitá vec, ktorú si tu treba uvedomiť → DEGRADÁCIA ŠTRUKTÚRY →spravím krásny návrh, kódnem brutal dobrú vec, zákazník mi odreportuje, že mu to crashlo pri nejakej situácii – tým, že tam na rýchlo spravím crash fix v časti, v ktorej taký kód nemá čo robiť si rozbíjam pôvodný krásny návrh a po čase mi z toho vznikne strašný bordel (prednášal aj marcus na VPPJ) → riešením je REFACTORING

softvérový proces

- definuje kto, čo, kedy robí a ako dosiahneme určitý cieľ
- určuje abstraktnú množinu činností, ktoré sa majú vykonať pri vývoji softvérového výrobku

softvérový projekt

- vykonanie činností definovaných v softvérových procesoch

vyspelosť softvérového projektu

- **CMMI** (z angl. Capability Maturity Model Integration) je model základných procesov v organizácii spolu s odporúčaniami ako ich efektívne implementovať. Pomáha zlepšiť prehľadnosť životného cyklu produktu, integrovať skúsenosti z odlišných oblastí, ...

činnosti vo vývoji softvéru

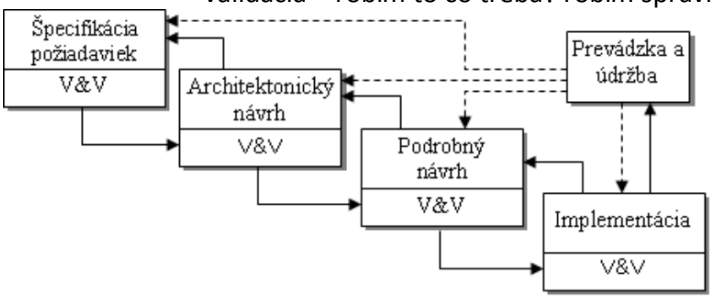
- analýza a špecifikácia požiadaviek
  - získavanie, analýza, definovanie a špecifikácia požiadaviek (čo chce používateľ)
  - štúdia vhodnosti – mala by byť rýchla a lacná
- architektonický návrh
  - návrh dekompozície systému
  - určenie vzťahov medzi časťami systému
  - špecifikácia funkcionality a ohraničení pre každý podsystém
- podrobný návrh
  - dokončenie podrobného návrhu vstupov a výstupov pre každý podsystém
  - návrh rozhrania pre každú súčiastku
  - návrh logickej a fyzickej štruktúry údajov
  - návrh algoritmov
- implementácia a testovanie súčiastok
  - programová realizácia softvérových súčiastok
  - vypracovanie dokumentácie k súčiastkam
  - testovanie implementovaných súčiastok
- integrácia a testovanie systému
  - spájanie súčiastok do systému (integrácia, najčastejšie inkrementálna)
  - dokončenie používateľskej dokumentácie
  - testovanie podsystémov a systému
- akceptačné testovanie a nasadenie
  - testovanie podsystémov a celého systému s cieľom preukázať používateľovi splnenie požadovaných vlastností
- prevádzka a údržba
  - zabezpečenie prevádzky systému
  - riešenie problémov s používaním softvéru
  - oprava, rozširovanie, prispôsobovanie softvéru podľa požiadaviek okolia

životný cyklus

- definuje jednotlivé etapy a pre každú z nich činnosti, ktoré sa majú vykonať, vstupy a výstupy
- rozdelenie (dekompozícia) zložitejšieho problému na jednoduchšie, ľahšie problémy
- model životného cyklu – definuje jednotlivé kroky, ich časovú následnosť, nedefinuje dĺžku a rozsah krokov, každá etapa je dobre definovaná, vie sa vyhodnotiť jej správnosť
- najrozšírenejšie modely: vodopádový, inkrementálny, iteratívny
- pre všetky podsystémy sa nemusí použiť ten istý model, napr. ak ide o podsystém so slabou špecifikáciou, použije sa iteratívny model spojený s prototypovaním; iný systém môže mať dobrú špecifikáciu, použije sa vodopádový model

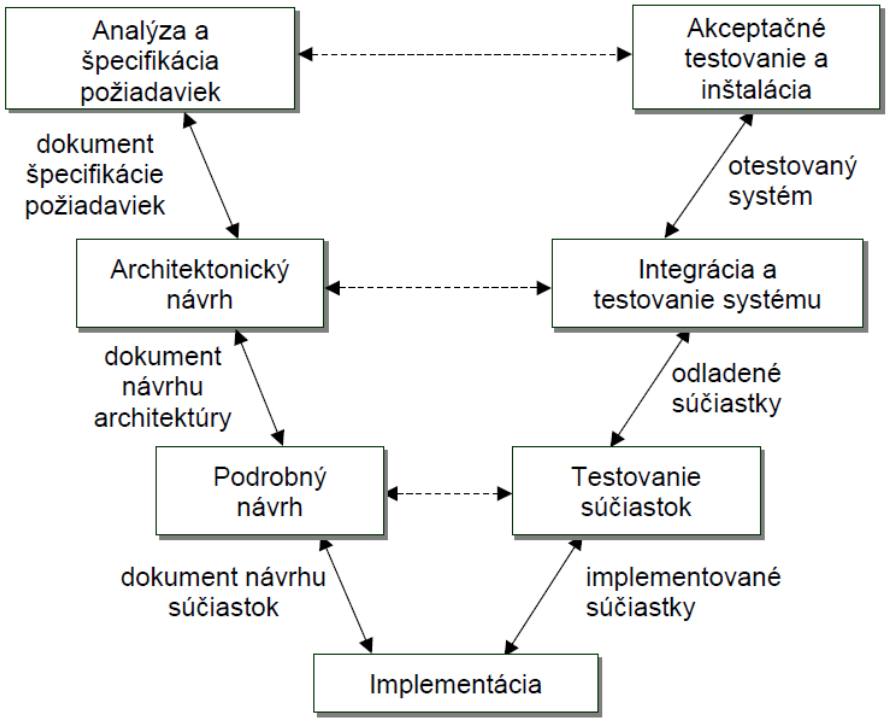
vodopádový model

- nasledujúca etapa sa začne vykonávať až po dokončení predchádzajúcej
- referenčný model, pôvodne predstavený ako chybný, nefungujúci model
- po každej fáze by sme si ňou mali byť 100% istý predtým ako prejdeme k ďalšej fáze
- používa sa v obrovských firmách – vývoj dobre zvládnutých aplikácií, známe postupy
- **V&V** – verifikácia a validácia – verifikácia = robím tak ako treba? robím výrobok správne?
  – validácia = robím to čo treba? robím správny výrobok?



V-model

- považovaný za typ vodopádového modelu, resp. jeho „rozšírenie“
- plné čiary zobrazujú následnosť krokov (aspekt následnosti)
- prerušované čiary spájajú časti na rovnakej úrovni abstrakcie (aspekt abstrakcie)
- ľavá vetva zobrazuje „čo a ako“, pravá vetva „ako sa to overuje“ – konkrétnu realizáciu



inkrementálny model (prírastkový)

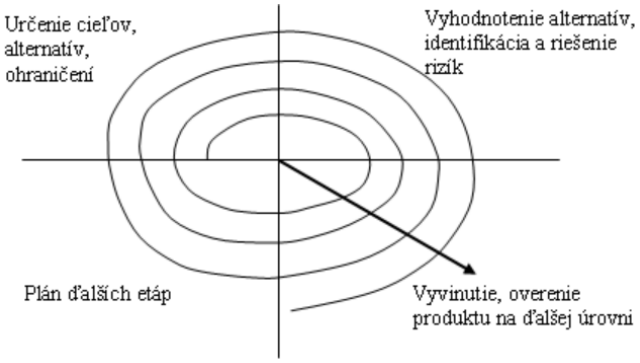
- najskôr sa identifikujú všetky požiadavky a navrhne sa architektúra systému (hrubá analýza)
- postupne sa pridávajú časti, ktoré sa stanovujú na základe špecifikácie celého systému
- systém sa vytvára a odovzdáva používateľovi po častiach

iteratívny model

- v každej iterácii sa vytvorí vykonateľný výsledok
- súbežne prebiehajú: návrh, implementácia, testovanie
- požiadavky sa všetky nedefinujú na začiatku (náčrt špecifikácie), využíva sa prototypovanie
- hodí sa ak nevieme dopredu špecifikáciu – evolučný vývoj, a pre systémy s „krátkym“ životom

špirálový model

- uvažuje aspekty manažmentu
- každé kolo špirály predstavuje etapu v procese tvorby softvéru
- zavádza analýzu rizík



WinWin model

- vyvinutý na základe špirálového modelu
- definuje množinu aktivít, ktoré smerujú k „výhre používateľa“ získaním produktu, ktorý zodpovedá jeho požiadavkám a „výhre vývojára“ vytvorením výsledku s reálnym rozpočtom a rozvrhom

komponentový model

- vývoj založený na znovupoužití softvérových súčiastok – ich vyhľadanie a integrácia

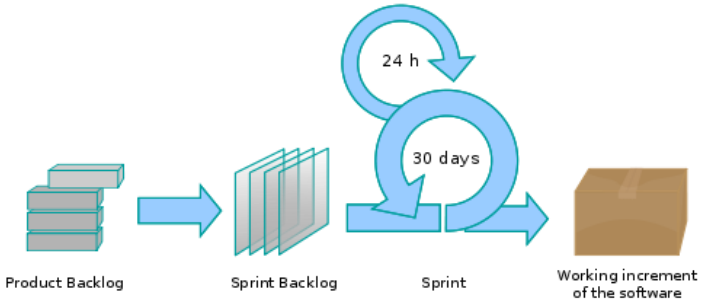
agilný vývoj softvéru

- skupina metodológií vývoja softvéru
- uprednostňuje:
  - *jednotlivca a interakciu* pred procesmi a nástrojmi
  - *fungujúci softvér* pred obsiahlou dokumentáciou
  - *spoluprácu so zákazníkom* viac ako rokovania o zmluvách
  - *reakciu na zmeny* viac ako dodržiavanie plánu
- hlavné charakteristiky:
  - inkrementálny a iteratívny vývoj
  - časté dodávky produktu
  - aktívna účasť používateľa na projekte
  - skoré a časté testovanie
  - vysoká kolaborácia a kooperácia všetkých zúčastnených
  - samo-organizované tímy (malý dôraz na manažérov)
- zahŕňa viacero metodológií (XP, Feature Driven Development, Scrum, ...)

scrum

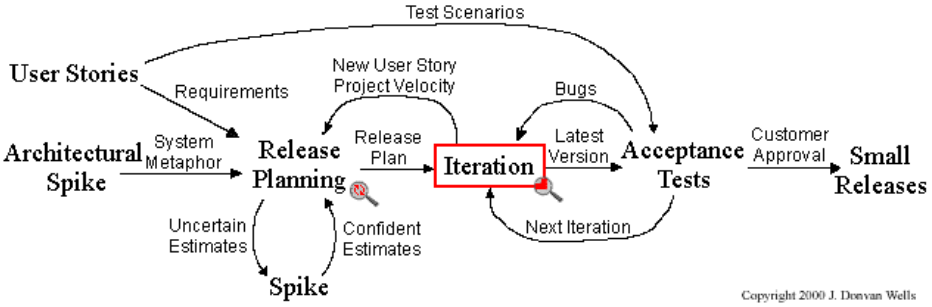
- agilná metodológia vývoja (dá sa aplikovať aj inde ako vo svete softvéru)
- definuje hlavné role (pigs):
  - Team – malá skupina ľudí, ktorí robia analýzu, dizajn, implementáciu aj testovanie
  - Scrum Master – chráni team od okolitých rušivých vplyvov (ale pozor, nevedie ho)
  - Product Owner – zástupca zákazníkov
- pomocné role (chickens):
  - Managers – manažéri, zabezpečujú prostredie a prostriedky na vývoj
  - Stakeholders – zákazníci

- proces vývoja:
    - product backlog
        - zoznam všetkého, čo kto chce do produktu (bugs, features)
        - Product Owner v ňom usporiada veci podľa priority
    - sprint backlog
        - zoznam vecí vybraných z vrchu product backlogu, ktorý si vytvára tím podľa toho, čo by mohli stihnúť v najbližšom šprinte
        - každý šprint má vždy rovnakú vopred dohodnutú dĺžku (týždeň, mesiac,...)
        - rozdelí sa na malé úlohy (tasks), ktoré sa potom vyvesia na viditeľné miesto
        - a každý si sám zvolí, na ktorej z nich chce práve pracovať. Keď ju dokončí, vyberie si ďalšiu.
        - na konci každého šprintu je sprint review, kde sa zhodnotí, čo sa urobilo
        - ešte aj každý deň je jeden daily scrum, čo je krátky meeting, kde môže každý povedať, čo urobil, čo plánuje robiť, alebo čo mu leží na srdci a pod ;-)



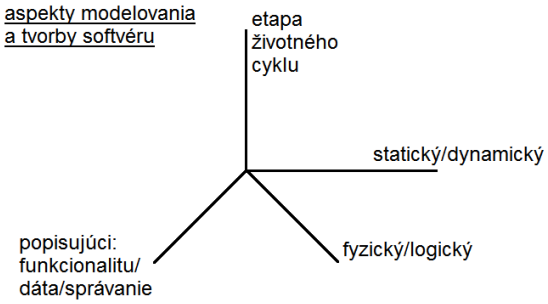
Extreme Programming (XP)

- agilná metodológia vývoja softvéru
- snaha znížiť náklady spojené s neustále sa meniacimi požiadavkami cez krátke vývojové cykly
- zdôrazňuje:
    - že kód je jediný skutočný produkt vývoja
    - častú integráciu
    - kolektívne vlastníctvo kódu
- používa:
    - iteratívny prístup
    - test driven development (TDD) – najskôr test, potom kód
    - pair programming
    - user stories – nie sú až také detailné ako use cases
    - akceptačné testy



model

= zjednodušená napodobenina originálu, na určitej úrovni abstrakcie
→ fyzický (AKO) / logický (ČO)
→ statický (elementy systému a vzťahy medzi nimi)/dynamický (správanie sa systému v čase)
→ podľa etapy život. cyklu → modely analýzy / návrhové modely / implementačné modely
– aspekty – akési dimenzie, podľa ktorých vieme zaradiť kam daný model patrí

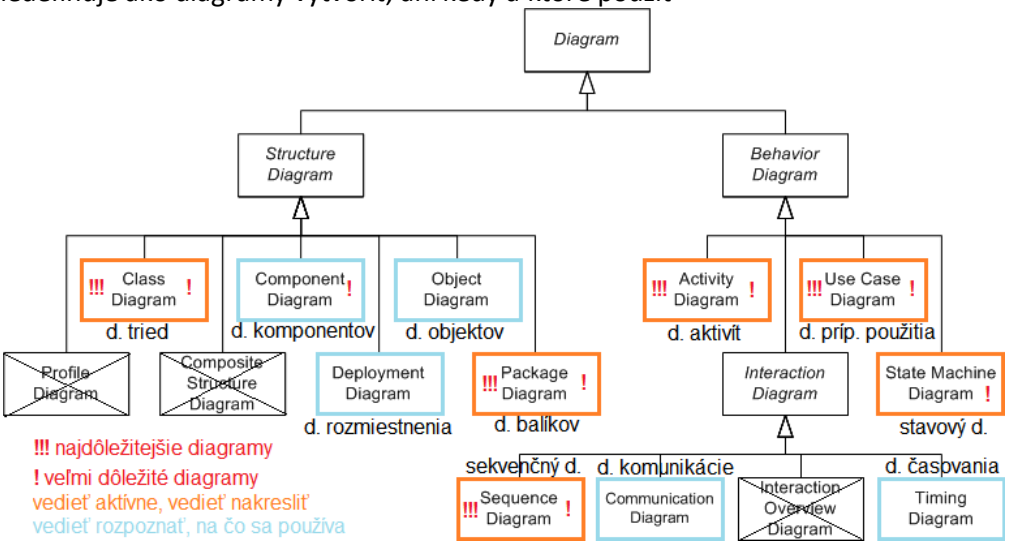


softvérový systém možno opísať z dvoch pohľadov:

- **štruktúrny model** (štruktúra, statické aspekty systému)
  - z tohto modelu sa samostatne opisujú:
    - model údajov
    - objektový model
- **model správania** (správanie systému v súvislosti s časom – dynamický model)
  - z tohto modelu sa samostatne vyčleňujú:
    - funkcionálny model

UML

- je jazyk na modelovanie s definovanou syntaxou a sémantikou
- snaží o zjednotenie jazykov na modelovanie; zjednocuje doterajšie osvedčené modely
- používa sa na vizualizáciu, špecifikáciu, konštrukciu a dokumentovanie softvérových systémov alebo ich častí
- definuje sadu modelov, ktoré sa používajú v procese tvorby softvéru
- nedefinuje ako diagramy vytvoriť, ani kedy a ktoré použiť



v ktorej etape ktorý UML diagram? (výsledok diskusie na fiitkarovi)

- analýza - diagram prípadov použitia, sekvenčný a diagram aktivít (dôležitá úroveň abstrakcie)
- návrh - diagram komunikácií, sekvenčný, tried, objektov, stavový, prípadov použitia, aktivít
- implementácia - diagram komponentov, rozmiestnenia a balíkov

mechanizmy rozširovania jazyka UML (dodefinovanie jazyka tak, aby vyhovoval špecific. potrebám)

- stereotypy
- ohraničenia (constraints)
- označené hodnoty (tagged values) - "značky"

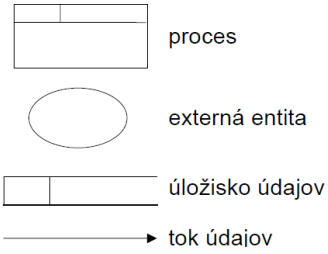
Funkcionálne modely

- softvérový systém chápe ako množina funkcií, resp. služieb, ktorými sa napĺňa požadovaný účel, resp. požadovaným spôsobom sa transformujú údaje
- data flow diagram (DFD), diagram prípadov použitia (use case)

DFD

- opisuje podstatu transformácie údajov, ktoré vstupujú do systému, vo vnútri systému menia svoju formu, uchovávajú sa v systéme, vystupujú zo systému
- systém sa modeluje ako sieť procesov, ktoré spracúvajú údaje v systéme
- uzly = úložiská údajov, procesy a vybrané objekty prostredia, v ktorom sa systém nachádza
- hrany = toky údajov

Gane-Sarson:

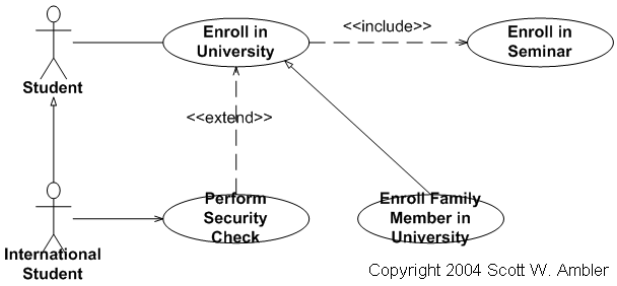


- proces = vyjadruje podstatu spracovania
  - procesy možno dekomponovať, DEKOMPOZÍCIA
- externá entita = opisuje zdroj alebo cieľ toku údajov
  - používateľ, organizácia, iný systém spracovania informácií,...
- úložisko údajov = vyjadruje obsah uchovávaných informácií
- tok údajov = reprezentuje informačný obsah daného toku údajov
- postupy pri tvorbe hierarchie DFD:
  - zhora nadol → zdola nahor → zhora nadol
  - zo stredu von
- DFD neznázorňuje riadenie, ani čas a časové následnosti, ani podrobnosti vzťahov medzi vstupmi a výstupmi
- DFD zachytáva tok údajov v systéme; postupnosť krokov algoritmu realizujúceho spracovanie
- DFD neumožňuje (ani sa o to nesnaží) zachytiť cykly a vetvenia pri opise spracovania

diagram prípadov použitia (use case diagram, UCD)

- interakcia používateľa a systému
- funkcionalita systému z pohľadu používateľa (uvažujú sa všetky spôsoby použitia systému)
- prípad použitia (use case, jedna bublinka) = činnosť, sloveso, interakcia
- hráč (actor, panák) = používateľ, systém, abstraktná vec (napr. čas)
- znázorňuje sa (vzťah «extend»):
  - normálny (štandardný) tok udalostí – takzvaný „happy-day“ scenár
  - regulárne varianty
  - zvláštne prípady
  - výnimočné, chybové prípady
- používa sa na: modelovanie požiadaviek na systém, analýzu scenárov, komunikáciu s používateľom, modelovanie kontextu systému, odhad času a úsilia, ktorý bude treba na tvorbu systému, stanovenie priorít, „synchronizácia“ jednotlivých modelov systému, pri testovaní systému (čierna skrinka)

- extend vs include
    - extend = nejaká činnosť **rozširuje** inú, vyjadruje činnosť, ktorá sa nemusí vždy vykonať, akási "optional" funkcionalita – čítaj „rozširuje“ v smere šípky
    - include = nejaká činnosť **zahŕňa** inú, väčšinou takú, ktorá sa vždy vykoná – viď. príklad, zapísanie na univerzite zahŕňa zapísanie predmetov



Model údajov

- systém sa chápe ako prostriedok na uloženie a opätovné získanie (transformovaných) údajov
- identifikovanie údajov, ktoré systém prijíma, s ktorými pracuje a ktoré produkuje, vyjadrenie vzťahov medzi identifikovanými údajmi a ich časťami, určenie obsahu údajov.
- dátová entita = akákoľvek informácia, ktorú treba uchovávať (podstatné meno, jednot. číslo)
- inštancia = špecifický, jedinečný výskyt entity
- entity sa odlišujú menom a zoznamom atribútov, inštancie sa odlišujú hodnotami atribútov
- vzťah /väzba medzi entitami: sloveso
- kvantifikácia vzťahu (kardinalita): stanovenie počtu výskytov určitej inštancie jednej dátovej entity pre jednu inštanciu druhej dátovej entity (1-1, 1-N, M-N)

entitno-relačný diagram (entity-relationship diagram, ERD)

- znázorňuje entity a vzťahy medzi nimi
- neznázorňuje vznik, modifikáciu a zánik údajových entít a ani tok spracovania údajov
- dátové entity ale aj vzťahy sú uzly, hrany sú len prepojenia medzi entitami a vzťahmi
- umožňuje definovať aj atribúty vzťahov

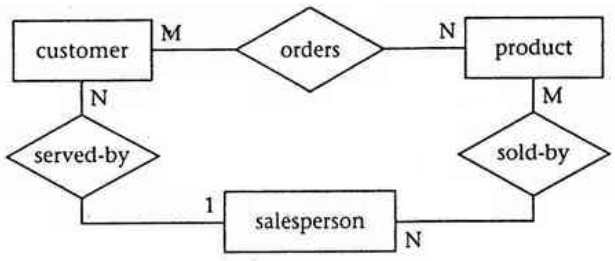


diagram modelu údajov (model diagram)

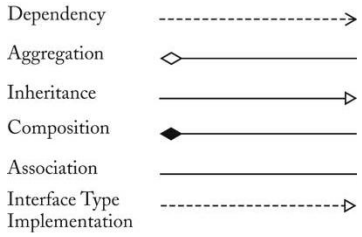
- vychádza z entitno-relačného diagramu
- vzťahy sa nemodelujú uzlami
- graf obsahujúci ako uzly dátové entity a hrany reprezentujú vzťahy nimi
- rôzne notácie na vyjadrenie kardinality
    - 0,1,* ako v RSA
    - 0,1,M,N
    - alebo tie také divné čiary, krúžky a vidličky (používa sa asi len tu)

diagram tried (class diagram)

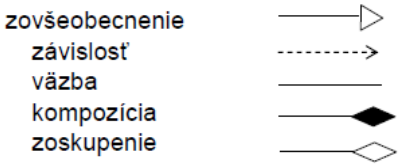
- najpoužívanejší a základný diagram v objektovo-orientovanej analýze a návrhu
- zachytáva štruktúru tried, rozhraní a spoluprác spolu s ich vzájomnými vzťahmi

Dependency	---------->	The source element depends on the target element and may be affected by changes to it
Association	——	The description of a set of links between objects
Aggregation	◇——	The target element is a part of the source element
Composition	◆——	A strong (more constrained) form of aggregation
Generalization	——▷	The source element is a specialization of the more general target element and may be substituted for it
Realization	----------▷	The source element guarantees to carry out the contract specified by the target element

OOP „flashback“



screen z materiálov:



- agregácia = „môže mať“, kompozícia = „vždy musí mať“

postup pri tvorbe:

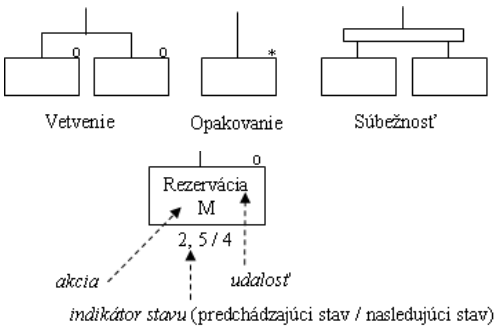
- 1. Identifikácia dátových entít – konkrétne veci, koceptuálne entity, udalosti, rozhrania
- 2. Definovanie dátových entít – skúma sa identita entity, jej účel, z čoho sa skladá (atribúty)
- 3. Určenie vzťahov medzi dátovými entitami – pomenovanie, analýza, kvantifikácia vzťahov
  – rozloženie vzťahov M ku N pomocou väzobných entít
- 4. Stanovenie atribútov
- 5. Normalizácia – s cieľom zlepšiť „performance“, spojenie toho čo sa opakuje (DRY), ...
- 6. Denormalizácia, distribúcia údajov - návrh fyzického modelu údajov, pričom sa uvažujú konkrétne parametre vyvíjanej aplikácie, efektívnosť, požiadavky na distribúciu údajov

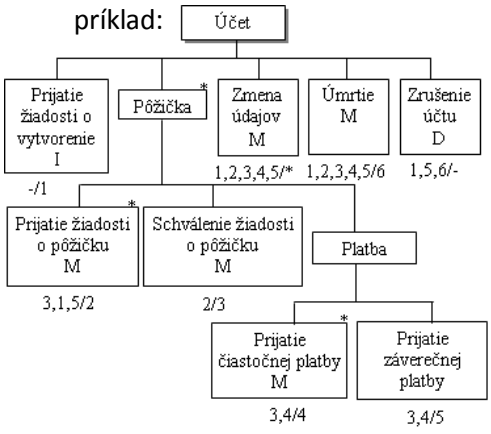
model správania

- zameriava sa na správanie systému v súvislosti s časom (dynamický model), sleduje sa ako systém reaguje v čase a ako sa mení jeho správanie, dopĺňa funkc. model a model údajov
- graf životného cyklu údajovej entity, stavový diagram (UML), diagram činností (UML),
- interakčné diagramy (diagram spolupráce, sekvenčný diagram, UML)

model životného cyklu dátovej entity

- opisuje časovú následnosť jednotlivých funkcií, ktoré pracujú s dátovou entitou
- všetky inštancie určitej dátovej entity majú rovnaký graf životného cyklu, každá inštancia sa však správa nezávisle a postupuje cez „niektoré stavy“ príklad:
- notácia:





stavový diagram (state transition diagram, statechart diagram)

- prostriedok na vyjadrenie kauzálnych a časových súvislosti (následnosti) akcií a stavov v systéme (časti systému, triedy, údajovej entity,...)
- na systém sa pozeráme ako na množinu stavov; systém reaguje na udalosti
- znázorňujú sa povolené prechody medzi stavmi, v ktorých sa môže daný objekt nachádzať
- stav = pomenovaná situácia objektu, väčšinou prídavné meno; množina okolností alebo atribútov, ktoré charakterizujú osobu alebo vec v danom časovom okamihu
- uzly sú stavy, hrany prechody medzi stavmi → udalosť [podmienka] / akcia

diagram činností (activity diagram)

- znázorňuje tok riadenia medzi činnosťami systému
- NIE JE to špeciálny druh stavového diagramu – platilo kedysi, už nie
- činnosť – predstavuje úlohu, ktorú musí vykonať systém alebo človek
- akcia – je špeciálny prípad činnosti, ktorá je elementárna, t.j. ďalej nedeliteľná
- ďalšie entity: rozhodovací blok (branch), spojenie (join), rozvetvenie (fork)
- používa sa na:
    - modelovanie tokov práce (workflow)
    - modelovanie operácií – v tomto prípade sú v diagrame akcie, nie činnosti

sekvenčný diagram (sequence diagram)

- modelovanie dynamického správania systému, podsystému, triedy, operácie
- znázorňuje interakcie medzi objektami v časovej postupnosti
- základné entity: objekt (inštancia triedy), správa
- ale tiež môže obsahovať: inštanciu rozhrania, komponentu, balíka, uzla.. poznámka, ohraničenie
- používa sa na modelovanie dynamického správania systému na všetkých úrovniach abstrakcie (konceptuálna, špecifikačná, implementačná) s dôrazom na usporiadanie v čase

diagram komunikácie (communication diagram)

- modelovanie dynam. správania pomocou interakcií medzi objektami a ich aktuálnych vzťahov
- sémanticky ekvivalentný so sekvenčným diagramom s takým rozdielom, že čas nie je samostatná dimenzia znázornená v grafe, ale poradie posielania správ sa vyjadruje ich číslovaním
- znázorňujú sa aj väzby (tak ako v diagrame tried)

princípy návrhu

- abstrakcia = uvažovanie o niečo bez podrobností ←→ konkretizácia = pridávanie podrobností
- zovšeobecnenie ←→ špecializácia
- dekompozícia – funkcionálna / orientovaná na údaje / orient. na udalosti / objektovo-orient.

  hierarchia: systém → podsystém → modul
- podsystém = systém, ktorého činnosť nezávisí od služieb, ktoré vykonávajú iné podsystémy
- modul = súčiastka systému, ktorá vykonáva jednu alebo viac služieb (funkcií) a používa pritom služby iných modulov – nepovažuje sa za samostatný systém

stratégie návrhu

- zhora nadol: systematická dekompozícia, identifikujú sa hlavné komponenty, postupné zjemňovanie (konkretizácia), hodí sa ak je špecifikácia požiadaviek jasná
- zdola nahor: postupne sa pridávajú vrstvy (zdola-nahor), hodí sa, ak už máme nejaký systém alebo súčiastky, ktoré možno znovu použiť
- kombinácia oboch prístupov: najčastejší prípad

modularita

- rozdelenie celku na menšie časti, s ktorými potom budeme vedieť jednoduchšie pracovať
- modulárny systém je taký, ktorý sa skladá z diskrétnych častí tak, že každú časť možno implementovať samostatne a zmena jednej časti má minimálny účinok na ostatné
- dekompozícia a abstrakcia pomáhajú dosiahnuť modulárnosť
- ciele modularity: nezávislý (samostatný) vývoj častí, zjednodušenie, opätovné použitie častí

vývojový diagram

- grafická reprezentácia riadenia procesu spracovania informácií
- hrany – tok riadenia; uzly – akcie a podmienky
- používa sa na zápis kostier procesov; rovnaká vyjadrovacia sila ako diagram činností

rozhodovacia tabuľka

- zoskupujú sa akcie s podmienkami, za ktorých možno tieto akcie vykonať
- deklaratívny prostriedok špecifikácie elementárneho procesu
- prostriedok modelovania správania
- možnosť hierarchického spájania rozhodovacích tabuliek; spojenie sa realizuje vložením "Prejdi na tabuľku..." medzi akcie v nadradenej tabuľke
- rozhodovacie pravidlo = stĺpec tabuľky – teda kombinácia podmienok a akcií, ktoré sa vykonajú, ak kombinácia podmienok má hodnotu pravda

rozhodovací strom

- sémanticky ekvivalentná technika s rozhodovacími tabuľkami
- rozhodovacie pravidlo = cesta od koreňa k listu

výhody a nevýhody rozhodovacích tabuliek a stromov

+ presný formát na vyjadrenie vzťahov medzi akciami a podmienkami
+ vytváranie logicky úplných špecifikácií
+ použitie na verifikáciu požiadaviek na procesy
+ použitie ako technická špecifikácia pre programátorov
+ možnosť hierarchického spájania rozhodovacích tabuliek
- náročné vytváranie zložitých tabuliek
- prešpecifikovanie požiadaviek (riešia sa kombinácie podmienok, ktoré sú síce logicky možné, ale z hľadiska analyzovaného systému nerelevantné)

testovanie (overovanie)

- statické – nevyžaduje vykonanie programu, v každej etape vývoja softvéru
    - prehliadka dokumentu – formálna (inspection) / neformálna (walkthrough)
    - matematická verifikácia – dôkaz, musí byť definovaná sémantika
- dynamické = proces odvodenia určitých vlastností výrobku na základe výsledkov použitia, prevádzky výrobku (vykonania softvérového systému) s vybranými vstupmi
  - techniky dynamického testovania:
    - náhodné testovanie
    - funkcionálne testovanie = black box, t.j. zadávajú sa vstupy, sledujú sa výstupy
    - štrukturálne testovanie = white box, na základe vnútornej štruktúry programu
    - testovanie rozhraní – na základe znalostí rozhraní medzi modulmi a špecifikácie

stratégie testovania

- testovanie zhora-nadol (bottom-up)
- testovanie zdola-nahor (top-down)
- sendvičové testovanie (sandwich)
  - kombinácia prvých dvoch; logické moduly sa testujú zhora-nadol, funkčné zdola-nahor
- jednofázové testovanie (big-bang)
  - moduly sa otestujú samostatne, potom sa naraz integrujú
- testovanie porovnávaním (back-to-back, comparision)
  - na tých istých vstupoch testujeme viac verzií toho istého modulu

alfa a beta testovanie

- alfa – na pracovisku, kde sa soft vyvíja, známe prostredie, testujú vývojári, používateľ
- beta – u používateľa, prostredie neznáme pre vývojový tím, testuje používatelia

typy zmien počas údržby

- oprava – odstraňovanie chyby (v dokumentácii, implementácii, návrhu, špecifikácii)
- prispôsobenie – zmeny v závislosti od zmien iných častí alebo zmien okolitého prostredia
- zlepšenie – vylepšenie o nejaké vlastnosti, ktoré neboli v špecifikácii pôvodného systému
- prevencia – modifikácia s cieľom zlepšenia ďalšej údržby
- speculative maintenance – periodická kontrola rôznych charakteristík, nový typ údržby pre webové aplikácie, kontrolujú sa odkazy

spätné inžinierstvo

- proces analýzy systému s cieľom identifikovať súčiastky systému a vzťahy medzi nimi a vytvoriť reprezentáciu systému v inej forme alebo na vyššej úrovni abstrakcie
- obnovenie návrhu (design recovery)
- obnovenie dokumentácie (redocumentation)
- problémy s nedostatkom informácií; predpoklady pri konkretizácii a zjemňovaní zvyčajne nie sú zdokumentované

reštrukturalizácia

- transformácia systému z jednej reprezentácie do inej, na rovnakej úrovni abstrakcie
- funkcia systému sa nemení

reinžinierstvo = spätné inžinierstvo + reštrukturalizácia

porovnanie metód vývoja softvéru

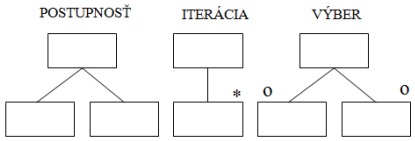
- spoločné:
  - základné princípy analýzy a návrhu
  - hierarchická reprezentácia systému (dekompozícia)
  - definujú postup
  - nedefinujú kritériá validácie
  - požiadavka na dôsledné uvažovanie rozhraní
- rozdielne:
  - stupeň formálnosti (formálne, semiformálne)
  - rôzne notácie
  - špecifický pohľad (funkcionálne, dátovo-orientované, objektovo-orientované)
  - špecifické postupy

štruktúrované metódy vývoja softvéru

- funkcionálne zamerané – projekt sa člení na malé, dobre definované činnosti; metódy určujú postupnosť týchto činností a súvislosti medzi nimi
- vychádza z princípu dekompozície (používajú sa hierarchické techniky ako napr. DFD)
- na pochopenie určitej časti danej úrovne dekompozície netreba znalosti o vnútornej štruktúre ostatných častí danej úrovne a častí nižších úrovní
- **SSADM** (Structured Systems Analysis and Design Method, Metodológia štruktúrovanej analýzy a návrhu systémov)
  - pokrýva etapy analýza, špecifikácia požiadaviek a návrh
  - analýza → štúdia vhodnosti → podrobná analýza → špecifikácia požiadaviek → návrh

dátovo-orientované metódy vývoja softvéru

- sústreďujú sa hlavne na štruktúru údajov
- hľadajú základnú štruktúru údajov a funkcionálna stránka sa odvodí od dátovej
- ide o funkcionálnu dekompozíciu, pričom sa vychádza z údajov
- **JSD** (Jackson System Development)
  - sústreďuje sa na návrh
  - štruktúru programu udáva štruktúra jeho vstupov a výstupov
  - hodí sa pre aplikácie, ktoré zahŕňajú spracovanie prúdu údajov (prekladače,...)



o**bjektovo-orientované metódy vývoja softvéru**

- systém sa chápe ako množina objektov a vzťahov medzi nimi, objekty komunikujú prostredníctvom správ
- každý objekt má svoj vlastný stav, na rozdiel od funkcionálneho prístupu, kde sa systém opisuje ako množina funkcií, ktoré zdieľajú spoločný stav
- základným konceptom je ukrytie informácií - navonok modulu (objektu) je viditeľné minimum informácií o vnútornom dianí v module (objekte), zapuzdrenie...
- dá sa povedať, že tu patrí aj UP (unified process, viď. ďalšia strana)

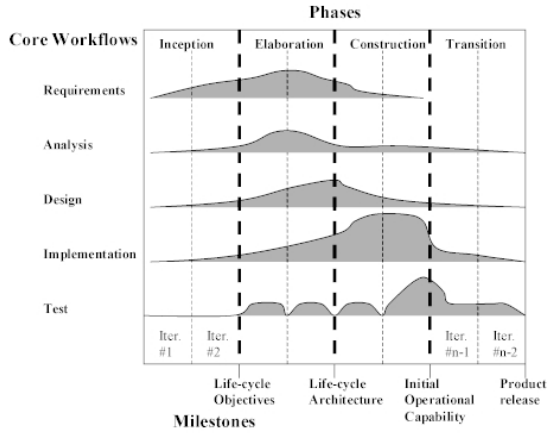
prototypovanie

- prototyp = "prvý z určitého druhu"
- softvérový prototyp = čiastočná implementácia systému, ktorej cieľom je dozvedieť sa niečo o riešenom probléme alebo o možnom riešení problému; orientuje sa na požiadavky
- požiadavky sú jasné, nejasné alebo neznáme
- prototypovanie pomáha redukovať tie nejasné a neznáme (kritické vzhľadom na návrh)
- prototyp na zahodenie (throw-away) – prototypujú sa nejasné pož. s cieľom porozumieť im
- evolučný prototyp (evolutionary) – prototyp. sa jasné pož. s cieľom spolupráce so zákazníkom

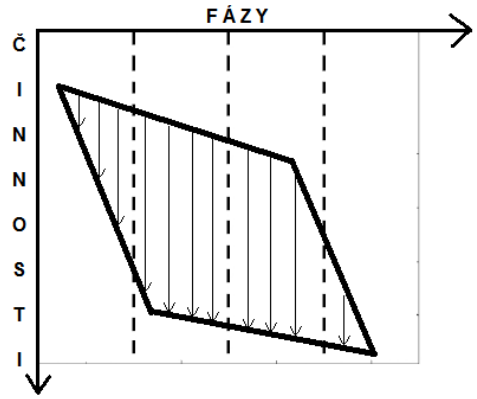
The Unified Process (UP, zjednotený proces)

- 1999 – Ivar Jacobson, Grady Booch, James Rumbaugh
- iteratívny a inkrementálny proces vývoja softvéru
- je vedený prípadmi použitia (uce case driven)
- je zameraný na architektúru (architecture centric)
- skladá za zo 4 fáz
    - zahájenie (inception)
    - rozpracovanie (elaboration)
    - vytváranie (construction)
    - nasadzovanie (transition)

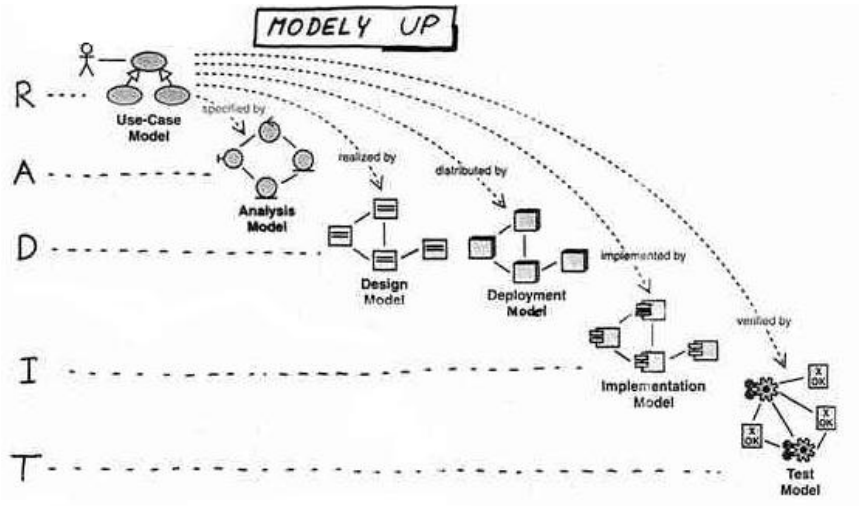
iteratívny vývoj:



schematické znázornenie:



znázornenie vývoja vedeného prípadmi použitia (use case driven development)
- na začiatku sú use casy, z ktorých sa vychádza pri analýze, návrhu, implementácii aj testovaní



Lehmanove zákony vývoja softvéru

1. zákon kontinuálnej zmeny – systém používaný v reálnom prostredí sa neustále mení pokiaľ nie je lacnejšie systém reštrukturalizovať alebo nahradiť novou verziou
2. zákon zvyšujúcej sa zložitosti – pri evolučných zmenách je program stále menej štrukturovaný a narastá jeho vnútorná zložitosť; odstránenie narastajúcej zložitosti vyžaduje dodatočné úsilie
3. zákon samoregulácie – rýchlosť zmien globálnych atribútov systému sa môže javiť v obmedzenom časovom intervale ako náhodná; pri dlhodobom pohľade sa však jedná o samoregulujúci proces, ktorý sa dá štatisticky sledovať a predvídať
4. zákon zachovania organizačnej stability – The average effective global activity rate in a system is invariant over product lifetime
5. zákon zachovania poznania – pri údržbe systému vznikajú nové a nové verzie; verzie, ktoré sú zmenené v nadpriemernej miere, majú sklon k zhoršeniu výkonnosti, spoľahlivosti, zvýšeniu chybovosti a prekročeniu časového a finančného rozpočtu
6. zákon kontinuálneho rastu – rýchlosť vývoja programu je približne konštantná
7. zákon klesajúcej kvality – kvalita systému s časom zdanlivo klesá pokiaľ systém nie je udržovaný a prispôsobovaný zmenám prostredia
8. zákon vplyvu spätnej väzby – evolution processes constitute multi-level, multi-loop, multi-agent feedback systems and must be treated as such to achieve significant improvement over any reasonable base