|  |
| --- |
| **Bible of SWING** |
| Vypracované zápisky z přednáškových slajdov |
|  |
| Študijný materiál k predmetu PB007: Software Engineering I. |
|  |
|  |
| **19.1.2013** |
|  |

Software process, UML language (1)

## Part 1

Software

* počítačové programy a asociovaná dokumentácia
* produkty môžu byť vyvíjané pre konkrétneho zákazníka alebo pre všeobecný trh

Atribúty dobrého SW

* požadovaná funkcionalita
* požadovaná výkonnosť
* spravovateľnosť (*maintainability*)
  + sw by mal byť vyvájaný takým spôsobom, že bude schopný odzrkadliť meniace sa potreby zákazníka
* spoľahlivosť (*dependability*)
  + zahŕňa *reliability*, *security* a *safety*
  + sw by nemal spôsobiť priamu fyzickú alebo ekonomickú škodu v prípade jeho zlyhania
  + neoprávneným užívateľom by nemal byť umožnený prístup k systému
* použiteľnosť (*usability*)
* využívanie zdrojov (*efficiency*)
  + sw by nemal plytvať systémovými zdrojmi (pamäť, cykly procesoru)
  + zahŕňa *responsiveness*, *processing time*, *memory utilization*
* prijateľnosť /prívetivosť (*acceptability*)
  + sw musí byť prijateľný pre cieľových užívateľov
  + mal by byť pochopiteľný, jasný a kompatibilný s inými systémami

SW inžinierstvo

* disciplína spájajúca všetky aspekty výroby softvérového produktu
* základné aktivity:
  + špecifikácia
  + vývoj
  + validácia
  + evolúcia

SW inžinierstvo vs. Systémové inžinierstvo

* System engineering
  + zaoberá sa všetkými aspektmi vývoja systému (HW, SW, procesné inž.)
* SW engineering
  + súčasťou Systémového inžinierstva

SW produkty

1. Generické produkty

* *stand-alone* systémy, ktoré sa predávajú hocijakému zákazníkovi
* napr. grafické programy, pomôcky pre správu menežmentu, computer-aided drafting software

1. Customizované produkty

* sw požadovaný špecifickým zákazníkom za účelom vyhovenia jeho konkrétnym potrebám
* napr. vstavané kontrolné systémy, systém pre správu vzdušnej dopravy

Aktivity SW procesu

* štruktúrovaná množina aktivít vykonaná s cieľom vývoja sw systému

1. špecifikácia: zákazník a inžinieri definujú vyvíjaný systém
2. vývoj
   1. analýza a dizajn: požiadavky sú pretavené do návrhu systémového dizajnu
   2. implementácia
3. validácia: overenie, že vyvinutý produkt splňuje zákazníkove požiadavky
4. evolúcia: modifikovanie sw na základe meniacich sa požiadavkov zákazníka a trhu

Modely SW procesu

* vodopádový model (waterfall model)
  + *plan-driven* model
  + oddelené a samostatné fázy špecifikácie a vývoju
* inkrementálny vývoj (incremental development)
  + plan-driven alebo agilný (*agile*)
  + špecifikácia, vývoj a validácia sú prekryté
* reuse-oriented model
  + plan-driven alebo agilný (*agile*)
  + systém je zložený z už existujúcich komponent

Waterfall model

* použitie
  + veľké systémy a generické produkty (plan-driven prístup zlepšuje koordinovanie práce)
* nevýhody:
  + neflexibilné členenie projektu do samostatných častí sťažuje reflektovanie meniacich sa požiadavkov zákazníka
  + model je vhodný pre prípady, kedy sú špecifikácie riadne pochopené a zmeny v priebehu vývoja budú maximálne limitované (avšak málo systémov má nemenné požiadavky)
  + lipnutie na dokumentácii

Inkrementálny vývoj

* výhody:
  + cena za implementovanie zmien požadovaných zákazníkom je redukovaná
  + jednoduchšia komunikácia so zákazníkmi – získavanie okamžitého feedbacku
  + rýchle doručenie a nasadenie použiteľných častí systému u zákazníka – zákazník využíva sw oveľa skôr, než je to umožnené za využitia vodopádového modelu
* nevýhody:
  + neviditeľnosť procesu –ťažké monitorovanie procesu manažérmi
  + systémová štruktúra má tendenciu sa s novými inkrementami znehodnocovať (potreba refactoringu, zmien v štruktúre – nárast nákladov)

## Part 2

Systémové modelovanie

* proces vytvárania abstraktných modelov systému – každý model reprezentuje inú perspektívu
* založené na grafickej notácii UML (*Unified Modeling Language*)

Perspektívy nahliadania na systém

* externá perspektíva
  + modelovanie systémových hraníc, kontextu, správania systému
  + UML diagramy: Use case diagram
* štrukturálna perspektíva
  + modelovanie organizácie systému alebo štruktúry dát, ktoré systém spracováva
  + Class diagram, Object diagram, Component diagram, Package diagram, Deployment diagram, Composite structure diagram
* perspektíva interakcie
  + modelovanie interakcií medzi systémom a jeho prostredím alebo medzi jednotlivými komponentmi
  + Sequence diagram, Communication diagram, Interaction overview diagram, Timing diagram
* perspektíva správania
  + modelovanie dynamického správania systému a toho, ako reaguje na udalosti
  + Activity diagram, State diagram

Populárne UML diagramy

* Use case diagram
  + modeluje interakcie medzi systémom a jeho okolím
* Class diagram
  + modelovanie tried systému a vzťahov medzi nimi
* Sequence diagram
  + modelovanie interakcií medzi užívateľmi systému (actors) a systémom, či medzi komponentami systému
* Activity diagram
  + modelovanie aktivít zahrnutých do procesu alebo spracovávania dát

Requirements engineering, UseCases (2)

## Part 1

Čo sú požiadavky?

* popisy systémových služieb a obmedzení, ktoré sú odhaľované počas procesu ich spravovania (tzv. requirements engineering process)
* forma je voliteľná – od vysoko abstraktnej špecifikácie obmedzenia až po detailnú špecifikáciu prostredníctvom matematickej funkcie

Typy požiadavkov

1. Užívateľské požiadavky

* písané v prirodzenom jazyku za využitia diagramov
* písané pre zákazníka

1. Systémové požiadavky

* štruktúrovaný dokument obsahujúci detailné popisy systémových funkcií, služieb a obmedzení
* definuje, čo má byť implementované

Funkčné a nefunkčné požiadavky

1. Funkčné požiadavky

* definície služieb, ktoré má systém poskytovať
* ako má systém reagovať na konkrétne vstupy
* ako sa má systém správať v konkrétnych situáciách
* definície toho, čo systém robiť nemá

1. Nefunkčné požiadavky

* vlastnosti a obmedzenia služieb (*timing*, *reliability*, *security* *constraints*)
* zohľadňujú systém ako celok, nie jeho individuálne časti či služby
* klasifikácia:
  + požiadavky na produkt (produkt musí dodávať istú kvalitu – rýchlosť, spoľahlivosť)
  + organizačné požiadavky (dôsledky politík a procedúr – použité štandardy)
  + externé požiadavky (vyvolané externými faktormi - legislatíva)

## Part 2

UML Use-cases

* technika založená na scenáriu, ktorá identifikuje užívateľov systému – tzv. actors v interakcii a ktorá túto interakciu aj sama popisuje

## Part 3

Requirements engineering

* iteratívna aktivita
* procesy:
  + výber požiadavkov a analýza
  + validácia požiadavkov (kontrola validity, konzistencie, úplnosti, reálnosti)
  + manažment požiadavkov

Stakeholderi

* koncoví užívatelia, manažéri, doménoví experti, obchodné spoločenstvá a únie
* napr.: pacienti, ktorých informácie spravuje systém; doktori, ktorí liečia pacientov; IT zamestnanci, ktorí inštalujú a udržujú systém
* problémy so stakeholdermi:
  + nevedia, čo v skutočnosti chcú
  + vyjadrujú požiadavky vo svojom vlastnom žargóne
  + konfliktné požiadavky
  + zmena požiadavkov – príchod nových stakeholderov, zmena prostredia

Prioritizácia požiadavkov

* MoSCoW kritéria
  + Must have – povinný požiadavok kladený na systém
  + Should have – dôležitý požiadavok, ktorý nemusí byť nutne zahrnutý do vývoja
  + Could have – voliteľný požiadavok
  + Want (Would) to have – nepotrebný požiadavok, ktorý môže počkať
* RUP atribúty
  + Status – Proposed/Approved/Rejected/Incorporated
  + Benefit – Critical/Important/Useful
  + Effort – number of person days/functional points/etc.
  + Risk – High/Medium/Low
  + Stability – High/Medium/Low
  + Target Release – future product version

## Part 4

UML: Use case actors

* hocičo v priamej interakcii s modelovaným systémom
* externé entity voči systému
* špecifikuje rolu, ktorú daná entita preberá počas interakcie so systémom

UML: Use cases

* prípad použitia systému špecifickým actorom
* vždy spustené actorom
  + primárny actor je triggerom
  + nula alebo viac sekundárnych actorov je v ďalšej interakcii s prípadom použitia
* happy day/perfect world scenario
  + prúd udalostí, ktorý je vykonávaný tak, ako bolo predpokladané a očakávané – bez chýb, odchýliek a prerušení

UML: events flow

* branching (keyword IF + Boolean expression)
* repetition (keyword FOR, keyword WHILE)
* alternatívne prúdy
  + nikdy sa nevracajú do hlavného prúdu
  + potenciálne veľa alternatívnych prúdov

Vzťahy medzi prípadmi užitia

* generalizácia actora
* generalizácia prípadu užitia
* <<include>>
  + pokiaľ prípady užitia zdieľajú chovanie, ktoré môže byť vyňaté do samostatnej inklúzie
* <<extend>>
  + rozširujúci prípad užitia vkladá chovanie do rozširovaného prípadu užitia

Non-functional Requirements Engineering (3)

## Part 1

Nefunkčné požiadavky

* spoľahlivosť
* výkonnosť
* modifikovateľnosť
* testovateľnosť
* použiteľnosť
* ...

Atribúty spoľahlivosti

1. Dostupnosť (Availability)

* pravdepodobnosť, že systém bude funkčný a zabezpečí požadované služby
* kvantitatívne vyjadriteľný: mean time to failure (MTTF)

1. Istota (Reliability)

* pravdepodobnosť, že systém nebude padať v danom prostredí a pre daný časový úsek
* kvantitatívne vyjadrený: probability of failure on demant (POFOD)

1. Zabezpečenie (Safety)

* vlastnosť reflektujúca schopnosť systému fungovať bez rizika spôsobenia zranenia či smrti človeku škody prostrediu
* terminológia:
  + riziko = pravdepodobnosť, že systém spôsobí nehodu
  + škoda = vyjadrenie straty spôsobenej nehodou

1. Bezpečnosť (Security)

* vlastnosť reflektujúca schopnosť systému ochrániť sa pred náhodným alebo zámerným útokom
* terminológia:
  + položka (asset) = niečo hodnotné chránené systémom
  + vystavenie (exposure) = možní strata alebo ujma systému
  + hrozba (threat) = okolnosti, ktoré majú potenciál vyústiť v stratu alebo ujmu

## Part 2

Aktivita

* aktivity sú siete uzlov spojené hranami (vyjadrujú prúd danej aktivity)
* kategórie uzlov:
  + Action nodes (action nodes, call nodes)
  + Control nodes (decision nodes, merge nodes, fork nodes, join nodes)
  + Object nodes
* kategórie hrán:
  + Control flows
  + Object flows

UML: Activity diagram

* token:
  + objekt, dáta – niečo, čo máme pod kontrolou
  + presun od zdrojového uzla k cieľovému uzlu za využitia hrán
* uzol sa aktivuje a vykoná, keď má na všetkých vstupných hranách tokeny a tieto zároveň splňujú preddefinované podmienky
* Action nodes:
  + na všetkých výstupných hranách ponúka token vtedy, keď je simultánne na všetkých vstupných hranách token a tieto splňujú preddefinované podmienky
* Decision nodes:
  + kontrolný uzol, ktorý má na vstupe jednu a na výstupe dve a viac hrán
  + zahrňuje vzájomne sa vylučujúce podmienky
* Merge nodes:
  + akceptuje jeden z viacerých alternatívnych prúdov
  + jeden a viac vstupných hrán, jedna výstupná hrana
* Fork nodes:
  + modelujú paralelný beh prúdov
  + tokeny z jednej vstupnej hrany sa replikujú do viacerých výstupných hrán
* Join nodes:
  + synchronizácia paralelných prúdov
  + dve a viac vstupných hrán a jedna odchádzajúca hrana

Analysis and Design (4)

## Part 1

Vývoj SW

* stupeň vo vývoji sw, kedy sa vytvára spustiteľný sw
* akcie:
  + Analýza (identifikácia procesov, entít, objektov a ich vzťahov)
  + Dizajn (doplnenie implementačných detailov)
  + Implementácia (proces zavedenia dizajnového návrhu do podoby programu)

Architektúrny dizajn a abstrakcia

1. architektúra v malom (analýza)

* zaoberá sa individuálnymi programami a ich dekompozíciou do komponent

1. architektúra vo veľkom (dizajn)

* zaoberá sa komplexným systémom, ktorý je tvorený programami a komponentami

## Part 2

SW systémy

* funkčne orientovaný pohľad
  + systém ako množina funkcií v interakcii
* dátovo orientovaný pohľad
  + systém ako množina dát a dátových štruktúr
* objektovo orientovaný pohľad
  + systém ako množina objektov v interakcii (dáta a operácie sú zapuzdrené)

Analýza

1. štruktúrovaná analýza

* delí projekt do malých, definovaných aktivít a definuje ich poradie a spôsob interakcie
* zvyšuje kvalitu systému
* metódy:
  + Yourdon: Modern Structured Analysis (YMSA)
  + Structured Systems Analysis and Design Method (SSADM)

1. objektovo orientovaná analýza

* modeluje systém ako skupinu objektov v interakcii
* objekt charakterizovaný triedou, stavom a správaním
* metódy:
  + Rational Unified Process (RUP)
  + Unified Process (UP)

## Part 4

Objekty

* pozostávajú z dát a funkcií, ktoré sú spolu zbalené do znovu použiteľnej jednotky
* každý objekt je instanciou triedy, ktorá definuje jeho vlastnosti (atribúty a operácie)

Zapuzdrenie

* dáta sú skryté vo vnútri objektu – prístup je umožnený len prostredníctvom operácie

## Part 5

CRC procedúra

* Class, Responsibilities and Collaborators

Object Oriented Analysis (5)

## Part 1

Terminológia

* spojenie (link)
  + spojenie medzi dvoma objektami
* asociácia (association)
  + vzťah medzi triedami
  + prítomná asociácia medzi triedami indikuje existujúce spojenie (link) medzi objektami týchto tried
  + spojenie (link) je instanciou asociácie takisto ako je objekt instanciou triedy

Asociačné triedy (Association classes)

* trieda vlastná danej asociácii
* instancie triedy sú spojenia (linky) ktoré majú atribúty a operácie

Kvalifikované asociácie (Qualified associations)

* redukujú *n-to-many* asociáciu *na n-to-1* asociáciu špecifikovaním unikátneho objektu z množiny

Závislosti (Dependencies)

* vzťah medzi dvoma elementmi, v ktorom zmena spôsobená jednému elementu vyvolá zmenu aj u druhého elementu
* typy:

1. Použitie (*Usage*) – use, call, parameter, send, instantiate
2. Abstrakcia (*Abstraction*) – trace, substitute, refine, derive
3. Povolenie (*Permission*) – access, import, permit

## Part 2

Generalizácia

* podtriedy (*subclasses*) dedia všetky vlastnosti nadtried (*superclasses*)
  + atribúty
  + operácie
  + vzťahy
  + stereotypy, tagy, obmedzenia
* podtriedy (*subclasses*) môžu pridávať nové vlastnosti
* podtriedy (*subclasses*) môžu prekrývať operácie nadtried (*superclasses*)
* instancia podtriedy (*subclass*) je použiteľná kdekoľvek je očakávaná instancia nadtriedy (*superclass*)

Abstraktné triedy a operácie

* abstraktné operácie = operácie, ktoré nie sú implementované
* z triedy, ktorá je tvorená abstraktnými operáciami, nemôže byť utvorená instancia

Polymorfizmus

* polymorfická operácia = operácia, ktorá môže byť implementovaná mnohými spôsobmi
* všetky konkrétne podtriedy (subclasses) musia implementovať abstraktné operácie nadtriedy (superclass)

Diagramy interakcie (Interaction Diagrams)

* zachytávajú interakciu v podobe:
  + lifelines – účastníkov v interakcii (atribúty: meno, selektor, typ)
  + správ (messages) – komunikácie medzi lifelines (synchrónne/asynchrónne)

Strutured Analysis (6)

## Part 1

Kontextový diagram (Context Diagram)

* tzv. model prostredia (*Environment model*)
* špeciálny prípad diagramu datového prúdu (*data flow*) obsahujúci jeden proces, ktorý reprezentuje celý systém
* kladie dôraz na:
  + ukončovacie prvky (terminators)
  + dáta (obdržané/vyprodukované)
  + datové sklady (data stores)
  + hranice systému

Diagram datového prúdu (Data flow Diagram)

* tzv. model správania (*Behavioral model*)
* grafická reprezentácia systému v podobe siete procesov, ktoré realizujú systémovú funkcionalitu a komunikujú prostredníctvom systémových dát
* elementy:
  + procesy – transformujú špecifické vstupy na výstupy
  + datové prúdy – modelujú cestu pre transfer dát medzi časťami systému
  + datové sklady – modelujú statickú kolekciu dát zdieľaných procesmi operujúcich v odlišnom čase
  + ukončovacie prvky – externá entita komunikujúca so systémom

## Part 2

Entity Relationship Diagram (ERD)

* modeluje štrukturálny pohľad s malou variáciou vzťahov
* obsahuje:
  + entity a ich typy
  + vzťahy a ich typy
  + atribúty a ich typy
* ERD sa zvyknú premietnuť na databázový model systému

Class Diagrams

* modelujú štrukturálne aj behaviorálne vlastnosti systému (atribúty a operácie)
* veľká variácia vtahov (asociácia, agregácia, kompozícia, závislosť, generalizácia)
* CD sa zvyknú mapovať na objekty z reálneho sveta

System Design (7)

## Part 1

Dizajnové vzory (Design patterns)

* izolovaný a popísaný *best practice* s ohľadom na špecifickú kvalitu atribútu
* spôsob využitia abstraktných znalostí o probléme a jeho riešení
* dôležitá je abstrakcia (použitie za rôznych okolností)
* príklady:
* Observer pattern
* Facade pattern
* Iterator pattern
* Decorator pattern

### Part 2

Zabezpečenie spoľahlivosti SW

1. Vyvarovanie sa chybám (Fault avoidance)

* snaha vyvarovať sa ľudským chybám a minimalizovať systémové chyby
* chyby sú detekované a opravené pred doručením zákazníkovi

1. Detekcia chýb (Fault detection)

* techniky verifikácie a validácie objavia a odstránia chyby pred nasadením

1. Tolerancia chýb (Fault tolerance)

* systém je navrhnutý tak, že chyby v SW nevyústia v systémové zlyhanie

Tolerancia chýb a taktiky obnovenia

* voting
  + procesy bežiace na redundantných procesoroch berú rovnaký vstup, vypočítajú výstupnú hodnotu, ktorá sa posiela na výber výsledku
* aktívna redundancia
  + redundantné komponenty reagujú na udalosti paralelne
  + použije sa hodnota len od jednej komponenty
* pasívna redundancia
  + jedna komponenta odpovedá na udalosti a informuje iné komponenty o stavových updatoch, ktoré musia urobiť
* náhrada (spare)
  + je konfigurovaná náhradná platforma, ktorá nahradí zlyhané komponenty
* shadow operation
  + zlyhaná komponenta beží krátku dobu v tieňovom móde, aby bolo zabezpečené zopakovanie chovania funkčných komponent predtým, ako sa znovu vráti do služby
* checkpoint/bod obnovy
  + generovanie konzistentného stavu, do ktorého sa môže komponenta/systém v prípade zlyhania vrátiť

### Part 3

Dizajnové guidelines

1. Vyvarovať sa tzv. *single point of failure*
2. Zabezpečenie v prípade pádu

* ochrana citlivých informácií pred neautorizovanými užívateľmi v prípade pádu systému

1. Logovanie užívateľských operácií

* vhodné na neskoršiu analýzu a monitorovanie chovania užívateľov

1. Nájsť rovnováhu medzi bezpečnosťou a použiteľnosťou
2. Dizajnovať obnoviteľný systém
3. ...

Schopnosť prežiť (Survivability)

* vlastnosť systému umožňujúca zabezpečiť elementárne služby aj v prípade, že je tento systém pod útokom alebo po narušení niektorej z jeho častí
* stratégie:
  + odolnosť (*Resistance*) – preferovať systém odolávajúci útokom
  + rozpoznanie (*Recognition*) – vedieť detekovať útoky a zlyhania
  + obnovenie (*Recovery*) – tolerovať problémy a ponúkať služby aj v priebehu útoku

### Part 4

Taktiky ovplyvňujúce výkon systému

* spotreba zdrojov
  + redukcia zdrojov potrebných na spracovanie prúdu udalostí
  + redukcia spracovávaných udalostí
  + kontrola využitia zdrojov
* správa zdrojov
  + súbežnosť spracovávania požiadavkov
  + udržiavanie kópie dát/výpočtov
  + navýšenie dostupných zdrojov
* vyšetrenie zdrojov
  + výber optimálnej stratégie rozdeľujúcej prístup k zdrojom

Taktiky ovplyvňujúce modifikovateľnosť systému

* lokalizácia zmien
* prevencia prenášaných zmien
* pozdržanie väzobnej doby

Taktiky ovplyvňujúce testovateľnosť systému

* správa vstupu/výstupu
* interný monitoring

### Part 5

Dizajnové triedy (Design classes)

* triedy, ktorých špecifikácie boli naplnené do takej miery, že môžu byť implementované
* anatómia dizajnovej triedy:
  + kompletná množina operácií (zoznam parametrov, návratových hodnôt, viditeľnosť, výnimky, gettery/settery, konštruktory a deštruktory)
  + kompletná množina atribútov (vrátanie typov a defaultných hodnôt)
* *well-formed* dizajnová trieda je (má):
  + kompletná a dostatočná
  + primitívna (operácie ponúkajú jednu, primitívnu atomickú službu)
  + minimálne previazaná (každá trieda modeluje jeden abstraktný koncept)
  + slabé väzby (asociácie len tam, kde je to naozaj potrebné)

Vnorené triedy (Nested classes)

* trieda definovaná vo vnútri inej triedy
* dostupné len cez obaľujúcu triedu a objekty obaľujúcej triedy

Architecture Design and Implementation (8)

## Part 1

Architektúrny vzor (Architectural pattern)

* štylizovaný popis dobrého dizajnového využitia, ktorý bol otestovaný v rôznych prostrediach
* príklady:
  + vzor vrstevnatej architektúry
  + vzor repozitárovej architektúry
  + vzor architektúry Client-server
  + vzor architektúry Pipe and filter

Vzor vrstevnatej architektúry

* používaný na modelovanie rozhraní podsystémov (*sub-systems*)
* organizuje podsystémy do vrstiev a každej z nich poskytuje služby
* zmena rozhrania vrstvy ovplyvní len susednú vrstvu

Vzor repozitárovej architektúry

* mechanizmus efektívneho zdieľania veľkého množstva dát medzi podsystémami

Vzor architektúry Client-server

* model distribuujúci dáta a ich spracovanie medzi množinou komponent
* *stand-alone* servery poskytujúce špecifické služby
* množina klientov využívajúca služby poskytované serverami
* siete sprostredkovávajú spojenie

Vzor architektúry Pipe and filter

* spracovanie vstupov na výstupy funkcionálnymi transformáciami
* nevhodný pre interaktívne systémy

## Part 2

Integrated development environments (IDE)

* množina softvérových nástrojov podporujúcich rozličné aspekty vývoja sw v rámci spoločného frameworku a užívateľského rozhrania

Open source licenčné modely

* The GNU General Public License (GPL)
  + pokiaľ použijete sw licencovaný pod GPL, daný sw musí byť open source
* The GNU Lesser General Public License (LGPL)
  + využitie open source kódu pod GPL bez nutnosti uvádzania zdroju
* The Berkley Standard Distributiou License (BSD)
  + žiadna povinnosť publikovať zmeny vykonané na open source kóde

## Part 3

Balíčky (Packages)

* logický mechanizmus na organizovanie modelovaných elementov do skupín
  + sémantické delenie elementov
  + definovanie sémantických hraníc

## Part 4

Component based development (CBD)

* konštruovanie sw z nahraditeľných *plug-in* častí
  + plug – poskytnuté rozhranie
  + socket – požadované rozhranie

Komponenta

* modulárna časť systému, ktorá zapuzdruje svoj obsah a ktorej prejavovanie sa je nahraditeľné v rámci jej rozhrania
* typy:
  + fyzická komponenta (instanciovaná počas behu – Enterprise JavaBean)
  + logická komponenta (podsystém)

Podsystémy (Subsystems)

* logický konštrukt, ktorý rozdeľuje väčší systém do spravovateľných časti
* nie sú instanciovateľné počas behu (ale ich obsah áno)

## Part 5

Artefakt

* reprezentuje typ konkrétnej veci použiteľnej v reálnom svete (napr. súbor)
* fyzická manifestácia jednej alebo viacerých komponent

User Interface Design (9)

## Part 1

WIMP paradigma

* *Windows*, *Icons*, *Menus* *and* *Pointing* *device*

Look & Feel prístup

* ovplyvňuje *experience* užívateľa a jeho identifikovanie sa s produktom
* *look* imitovateľný ľahko (farby, tvary), *feel* nie (dynamické správanie)

## Part 6

Stavové stroje (State machines)

* používané na modelovanie dynamického správania tried, prípadov užitia, podsystémov
* typy:
  + stavové stroje chovania (*Behavioral state machines*)
  + protokolové stavové stroje (*Protocol state machines*)
* udalosti v stavových strojoch:
  + *call* – volanie na vykonanie operácie
  + *signal* – balíček informácie poslaný asynchrónne medzi objektami
  + *change* – akcia vykonaná keď sa Boolean výraz zmení z false na true
  + *time* – vykonajú sa keď nadobudne časový výraz pravdivostnú hodnotu

Testing, Verification and Validation (10)

## Part 1

Testing

* demonštrácia toho, čo program dokáže a na objavenie programových chýb
* odhalí prítomnosť chýb, nie ich neprítomnosť
* cieľe:
  + demonštrovať developerovi a zákazníkovi, že sw spĺňa požiadavky (*validation testing*)
  + odhaliť situácie, v ktorých sa sw správa nekorektne/ nespĺňa požiadavky (*defect testing*)

Verifikácia vs validácia

* verifikácia
  + kontrola s účelom zabezpečenia konzistencie s danou špecifikáciou
  + Je systém tvorený správne?
  + Zhoduje sa so špecifikáciou?
* validácia
  + Vytvorili sme správny systém?
  + Spĺňa to, čo si zákazník prial?

Testovanie vs inšpekcia

* sw inšpekcia
  + statická verifikácia
  + analýza statickej systémovej reprezentácie s cieľom odhaliť problémy
* sw testovanie
  + dynamická verifikácia
  + vykonávanie programu a pozorovanie správania

## Part 2

Statická analýza

* verifikačná technika, ktorá nevyžaduje spútanie programu
* typy:
  + formálna verifikácia
  + model checking
  + automatizovaná analýza

Formálna verifikácia

* použitie s matematickou špecifikáciou produktu
* možné využiť v rôznych stupňoch vývojového procesu
* výhody:
  + tvorba matematickej špecifikácie vyžaduje detailnú analýzu požiadavkov – vedie k odhaleniu chýb
  + súbežné systémy môžu byť analyzované na *race condition* a vyvarovať sa *deadlocku*
  + detekcia implementačných chýb ešte pred testovaním
* nevýhody:
  + vyžaduje špecializované notácie pochopiteľné len doménovými expertmi
  + finančne náročná metóda
  + dôkaz môže obsahovať chyby

Model checking

* vyjadrenie systému v podobe konečne stavového modelu
* kontrola všetkých ciest a validovanie špecifikovanej vlastnosti

Automatizovaná analýza

* statické analyzátory na spracovanie zdrojového textu
* parsovanie programu a odhalenie potenciálne chybového stavu
* úrovne:
  + charakteristická kontrola chybovosti (tradičné chyby voči programovaciemu jazyku)
  + užívateľom definovaná kontrola chybovosti
  + kontrola assertov

## Part 3

Testovanie

1. Vývojové testovanie (Development testing)

* systém je testovaný počas vývoja

1. Release testing

* testovanie kompletnej verzie programu pred sprístupnením užívateľom

1. Užívateľské testovanie (User testing)

* užívatelia systému testujú systém vo svojom prostredí

Development testing

* Unit testing
  + testovanie samostatných jednotiek/funkcií/metód
  + nezohľadňuje sa ich implementácia (*black-box testing*)
* Component testing
  + testovanie jednotiek, ktoré sú integrované do podoby zloženej komponenty
  + testovanie rozhraní komponent
* System testing
  + testovanie komponent, ktoré sú integrované do celku – systému
  + testovanie interakcie komponent

User testing

* Apha testing
  + užívatelia spolupracujú s developerským tímom na testovaní sw v prostredí developerov
* Beta testing
  + užívateľom je krátkodobo sprístupnená plná verzia systému na experimentovanie a testovanie
* Acceptance testing
  + zákazníci testujú systém a rozhodujú sa, či môže byť nasadený medzi bežných užívateľom

Automatizované testovanie

* využitie automatizovaných testovacích *frameworkov* (JUnit)
* komponenty:
  + *setup part* (inicializácia systému, identifikácia vstupov a očakávaných výstupov)
  + *call part* (volanie metódy, ktorá je testovaná)
  + *assertion part* (porovnanie výsledku s očakávaným výsledkom)

Test-driven development

* prístup k vývoju sw, v ktorom sa prekrýva testovanie a vývoj samotného kódu
* testy sú písané ešte predtým, ako je napísaný kód
* kód sa píše inkrementálne spolu s testom pre daný inkrement
* výhody:
  + pokrytie kódu testami
  + regresné testovanie
  + zjednodušenie debuggovania
  + testy sú formou systémovej dokumentácie

Operation, Maintenance and System Evolution (11)

## Part 1

Zmena sw

* vývoj sw je nevyhnutný
  + objavujú sa nové požiadavky
  + zmena business prostredia
  + oprava chýb/bugov
  + nové hw vybavenie a zabezpečenie kompatibility

Fázy životnosti sw

1. Evolúcia (Evolution)

* sw je operatívny, vyvíja sa a sú doň implementované nové požiadavky

1. Obsluha (Servicing)

* sw ostáva použiteľný, ale aplikujú sa len zmeny, ktoré ho udržia v použiteľnom stave (*bug fixes*)
* nepridáva sa nová funkcionalita

1. Phase-out

* sw môže byť stále používaný, ale už sa v ňom nedejú žiadne zmeny

## Part 3

Údržba sw

* oprava softvérových chýb
* úprava sw na adaptovanie v inom operačnom prostredí
* úprava s cieľom modifikovať systémovú funkcionalitu

Cena údržby sw – čo ju ovplyvňuje?

* stabilita tímu
  + udržiavanie rovnakých zamestnancov počas celého vývojového procesu
* vzájomná zodpovednosť
  + vedenie vývojárov k tomu, že bude nutné v budúcnosti sw udržiavať
* schopnosti zamestnancov
  + tím údržby má často obmedzené doménové znalosti
* štruktúra kódu a jeho staroba
  + s vekom kódu degraduje jeho štruktúra a schopnosť adaptovať sa na zmeny

Systémové re-inžinierstvo (re-engineering)

* reštrukturalizácia alebo prepísanie časti systému bez zmeny jeho funkcionality
* výhody:
  + redukovanie rizika
  + redukovanie ceny (menej nákladné než vývoj nového sw)
* aktivity:
  + preklad zdrojového kódu do nového programovacieho jazyka
  + reverzné inžinierstvo – analýza programu s cieľom pochopiť ho
  + zlepšenie programovej štruktúry – reštrukturalizácia
  + modularizácia programu – reorganizovanie programovej štruktúry
  + *reengineering* dát – upratanie a reštrukturalizácia systémových dát

Refaktoring

* proces vykonávania zmien na programe s cieľom spomaliť jeho degradáciu s narastajúcim časom
* modifikovanie v podobe zmeny štruktúry, zjednodušenia a zvýšenia prehľadnosti
* *triggery* refaktoringu:
  + duplicitný kód
  + dlhé linkované metódy
  + switch výrazy
  + zhlukovanie dát
  + špekulatívna generalita

Software Development Management (12)

## Part 1

Prototypovanie sw

* prototyp je iniciálna verzia systému používaná na demonštrovanie konceptov a skúšanie nových možností

Inkrementálny vývoj

* rozbitie vývoja a doručenia na inkrementy – s každým inkrementom sa doručí časť požadovanej funkcionality
* prioritizácia užívateľských požiadavkov – tie s vyššou prioritou sú uprednostňované
* vyvíjanie konkrétneho inkrementu znamená zastavenie prijímania požiadavkov pre tento inkrement
* výhody:
  + zákazník má k dispozícii funkčné jednotky systému skôr
  + znižovanie rizika neúspechu projektu
  + služby s najvyššou prioritou bývajú otestované v najväčšom meradle
* nevýhody:
  + obtiažna identifikácia základných komponent, ktoré využíva každá časť vyvíjaného systému
  + iteratívne procesy si zakladajú na tom, že je špecifikácia vyvíjaná zarovno so sw – niektoré firmy s tým nemusia súhlasiť

Agilné metódy

* zamerané viac na kód než na dizajn
* založené na iteratívnom prístupe vývoja sw
* rýchle doručenie jednotiek funkčného sw
* adaptabilita na meniace sa požiadavky zákazníka
* limitovanie dokumentácie
* problémy:
  + udržať zákazníka zainteresovaného a v roli účastníka procesu vývoja
  + vývojársky tím neakceptuje princípy agilného vývoja
  + prioritizovanie požiadavkov v prípade viacerých *stakeholderov*
  + udržiavanie jednoduchosti vyžaduje prácu naviac

Plan-driven vývoj

* založený na oddelených vývojových fázach s tým, že na konci každej je produkovaný výstup tak, ako bol naplánovaný
* využíva iterácie

## Part 2

Manažment sw

* aktivity:
  + plánovanie projektu
  + manažment rizík
  + manažment ľudí
  + reporting
  + vyjednávanie kontraktov