

1. SYSTEM AND SOFTWARE ENGINEERING

Softvérové inžinierstvo sa zaoberá teóriami, metódami a nástrojmi pre profesionálny softvérový vývoj. Softvérové inžinierstvo sa zaoberá cenovo - efektívneemu vývoju softwaru.

Čo je software ?

Počítačové programy a súvisiaca dokumentácia, ako požiadavky, dizajn modelov a používateľských príručiek. Softwarové produkty môžu byť vytvorené pre konkrétneho zákazníka alebo môžu byť vyvinuté pre trh všeobecne.

Softwarové produkty môžu byť

- Generic - vyvinutý byť predané rôznym zákazníkom, napr. PC softvér, ako je Excel alebo Word.
- na zákazku (custom) - vyvinutý pre jediného zákazníka podľa jeho špecifikácií.

What is software engineering?

Softvérové inžinierstvo je inžinerska disciplína, ktorá sa zaoberá všetkými aspektmi softvérovej produkcie. Softvéroví inžinieri by mala prijať systematický a organizovaný prístup k práci a využiť vhodné nástroje a postupy v závislosti od problému ktorý treba vyriešiť, vývojove obmedzenia a dostupne zdroje.

Why is software engineering important?

Softvér musí byť spoľahlivý, bezpečný, použiteľný a udržiavateľný. Softvérové inžinierstvo sa výslovne zameriava na poskytovanie softvéru s týmito atribútmi. Softvérové inžinierstvo je obzvlášť dôležité pre systémy, na ktorých sú ľudia a firmy závislé, a ktoré sa používajú mnoho rokov.

What is the difference between software engineering and computer science?

Počítačová veda sa zaoberá teóriou a základnými princípmi, softvérové inžinierstvo sa zaoberá praktickosťou vývoja a poskytovaním užitočného softvéru.

What is system engineering?

Systémové inžinierstvo sa zaoberá všetkými aspektmi počítačového systému, vývojom, vrátane hardvéru softvéru a process inžinierstvo. Systémoví inžinieri sa podieľajú na špecifikácii systému, architektonické navrhovanie, integráciu a použitím v praxi.

What is a software process?

Súbor činností, ktorých cieľom je rozvoj alebo vývoj softvéru.

Všeobecné činnosti vo všetkých softvérových procesov sú:

- Špecifikácia - čo by mal systém robiť a jeho vývojove obmedzenia
- Vývoj - výroba softvérového systému
- Validácia - overenie, že softvér je to, čo zákazník chce
- Evolúcia - zmena programu v závislosti na meniacich sa požiadavkách.

What is a software process model?

Zjednodušená reprezentácia procesu softvéru, vychádzajúca (vyplývajúca) z určitého špecifického pohľadu.

Príklady procesných hľadísk sú

- Workflow perspektíva - sled činností;
- Dátový tok perspektíva - tok informácií;
- Role (Uloha) / action perspektíva - kto čo robí.

Všeobecné procesné modely

- Waterfall;
- Iterative development;
- Component-based software engineering.

What are the costs of software engineering?

Zhruba 60% nákladov sú náklady na vývoj, 40% sú náklady na testovanie. Rozdelenie nákladov závisí na vývoji modelu, ktorý sa používa.

What are software engineering methods?

- Model descriptions - Popisy grafických modelov, ktoré by mali byť vyrobené;
- Rules - Obmedzenia aplikované na modely systémov;
- Recommendations - rady pre zlepšenie praktizovania designu
- Process guidance (poradenstvo, odporúčania) – ktoré aktivity budú nasledovať

What is CASE (Computer-Aided Software Engineering) ?

CASE systems - sú často používané pre metódu podpory.

Upper-CASE - Nástroje na podporu skorého process activities

Lower-CASE -nástroje na podporu neskoršej činnosti, napr. programming, debugging, testing

What are the attributes of good software?

Maintainability(udržiavateľnosť) - Softvér musí vyvinúť na uspokojenie meniacich sa potrieb;

Dependability (spoľahlivost) - Softvér musí byť dôveryhodný;

Efficiency (účinnosť)- Softvér by nemala mať nehospodárnemu využívaniu systémových zdrojov;

Acceptability (akceptovateľnosť) - musia byť zrozumiteľné, použiteľné a kompatibilné s inými systémami.

KEY POINTS

Softvérové inžinierstvo je inžinierska disciplína, ktorá sa zaobrá všetkými aspektmi softvéru výroby. Softvérové produkty sa skladajú z vytvorených programov a súvisiacou dokumentáciou. Základné atribúty produktov sú udržiavateľnosť, spoľahlivosť, efektívnosť a použiteľnosť. Softvér proces sa skladá z činností, ktoré sa podieľajú na rozvoji softvérových produktov. Základné činnosti sú softvér, špecifikácia, vývoj, validáciu a evolúcia.

Critical Systems

Safety-critical systems - Porucha má za následok straty na životoch, zranenia, dopad na ziv. pros
Napr. Chemical plant protection system

Mission-critical systems - Failure results in failure of some goal-directed activity;
Napr. Navigačný systém vesmírnej lode

Business-critical systems – porucha spôsobí vysoké ekonomicke straty

Vývoj metód (Development methods) pre kritické systémy

Náklady zlyhania Critical systems sú také vysoké, že Development methods môžu byť použité tak že nie sú cenovo výhodné pre iné typy systémov.

Príklady metód rozvoja

- Formálne metódy vývoja softvéru
- Statická analýza
- Externé zabezpečenie kvality

Socio-technical critical systems

Hardware failure – hardware zlyha z dôvodu zleho navrhу alebo výrobnej chyby, skončila životnosť

Software failure – software zlyha kvôli chybe v jeho specifikácii, navrhу alebo implementácii

Operational failure (operacna chyba) – ludky faktor(chyba), stava sa najcastejšie

Dependability

Spolahlivý system je taky system ktoremu doveruju jeho pozívateľia. Hlavné okruhy spoľahlivosti sú: dostupnosť, spoľahlivosť(reliability), ochrana, bezpečnosť, chybova tolerancia, opraviteľnosť, udržovateľnosť

Maintainability

Ako co najjednoduchšie opraviť systém po najdení chyby, alebo upraviť systém s novymi funkiami(features), veľmi dolezite pre Critical systems z dôvodu udržby systémov

Survivability

Schopnosť systému nadálej poskytovať svoje služby používateľom, aj napriek tomu že na system mozu utocit. Je to stále čoraz dôležitejšia atribút distribuovanych systémov, ktorých bezpečnosť môže byť ohrozená.

Dva dôvody pre Dependability cost:

- Coraz drahsie vyvojove techniky pre dosiahnutie vyssej urovne spolahlivosti
- Zvysenie testovania a validacie systemu

Spolahlivosť (reliability)

• pravdepodobnosť bezporuchovej prevádzky systému na určitú dobu v danom prostredí pre daný účel

Dostupnosť(availabilty)

• Pravdepodobnosť, že systém, v okamihu, bude v prevádzke a je schopný dodávať požadované služby

Obe z týchto atribútov mozu vyjadrovať **kvantitatívnosť**.

System failure - Udalosť, ktorá sa vyskytuje v určitom okamihu, kedy systém neposkytuje služby podľa očakávaní jeho používateľom

System error - Chybný stav systému, ktorý môže viesť k takému systémovému správaniu, ktoré bude pre používateľa neočakávané

System fault- charakteristika systému ktorá moze viesť ku System error. Napríklad zlyhanie inicializacie premennej, ktore moze viesť k tomu že premenna bude mať zlu hodnotu.

Safety

Bezpečnosť je vlastnosť systému, ktorá odráža schopnosť systému pracovať, normálne alebo abnormálne, bez nebezpečenstva spôsobenia zranenia alebo usmrtenia človeka a aby nedošlo k poškodeniu životného prostredia.

Security

schopnosť ochrániť systém pred náhodným alebo zámerným vonkajším útokom

Damage from insecurity

- Denial of service (Odmietnutie služby) Systém je nútený do stavu, kedy bežné služby nie sú k dispozícii alebo tam, kde poskytovanie služieb je výrazne degradované
- Corruption of programs or data - Programy alebo údaje v systéme môžu byť zmenené neautorizovaným spôsobom
- Disclosure of confidential information (Zverejnenie dôverných informácií) - informacia ktora je riadená systémom može byť ukazaná ľuďom, ktorí niesu autorizovaní, nemajú opravnenie čítať alebo používať túto informáciu

KEY POINTS

Critical system je systém, kde porucha môže viest' vysokej ekonomickej strate, fyzickému poškodeniu alebo ohrozeniu života.

Dependability v systéme odráža to ako užívateľ verí systému.

Availability(Dostupnosť) systému je pravdepodobnosť s ktorou bude systém k dispozícii pre poskytovanie služieb na požiadanie

Reliability(spolahlivosť) systému sa pravdepodobnosť s ktorou systémové služby budú poskytované, ako je uvedené

Spolahlivosť a dostupnosť sú všeobecne považované za nevyhnutné, ale nie dostatočné pre safety and security.

Softvérové Procesy

Ciele

- Predstaviť model softvérového procesu
- Opísat tri generické modely procesov a kedy by sa mali používať
- Opísat ///////////////
- Vysvetliť Rational Unified Process model
- Predstaviť CASE technológiu kvôli podpore aktivít softvérových procesov

Softvérové procesy

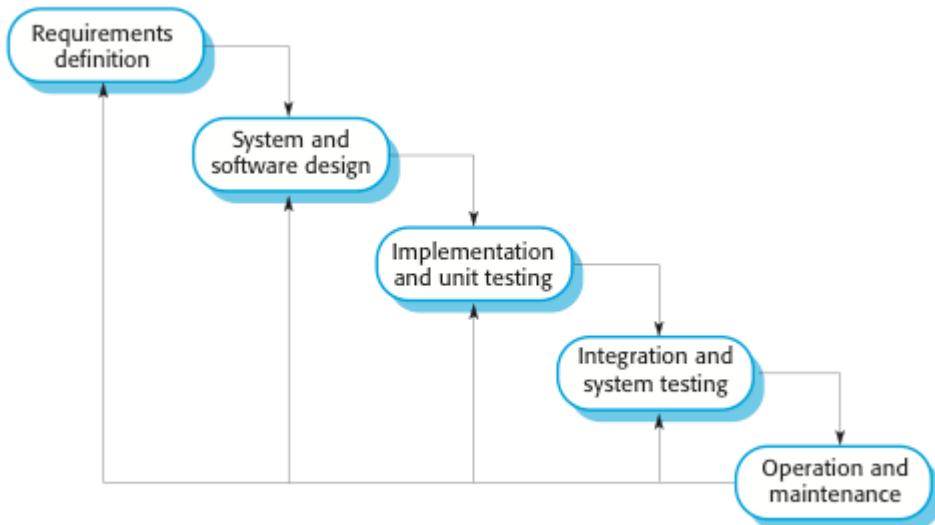
- Štruktúrovaná množina aktivít potrebná na vývoj softvérových systémov
 - o Špecifikácia
 - o Dizajn
 - o Overenie
 - o Evolúcia
- Model softvérového procesu je abstraktná reprezentácia procesu .
Predstavuje popis procesu z nejakých špecifických perspektív.

Generický model procesov

- Vodopádový model
 - o Oddeliť a odlišiť fázy špecifikácie a vývoja.
- Evolučný vývoj
 - o Špecifikácia, vývoj a overenie sú vrstvené (interleaved)
- Na existujúcich komponentoch založené softvérové inžinierstvo
 - o Systém je zostavený z existujúcich komponentov.

- Existuje mnoho variantov týchto modelov napr. formálny vývoj, kde sa použije proces podobný vodopádu ale špecifikácia je formálna špecifikácia, ktorá je upresnená cez niekoľko stupňov do implementovateľného dizajnu.

Vodopádový model



Fázy vodopádového modelu

- Analýza a definícia požiadaviek
- Dizajn systému a softvéru
- Implementácia a testovanie jednotiek
- Integrácia a testovanie systému
- Prevádzka a údržba
- Hlavnou nevýhodou tohto vodopádového modelu sú ťažkosti pri zmenách už keď bol proces spustený. Jedna fáza musí byť hotová pred posunutím sa na ďalšiu.

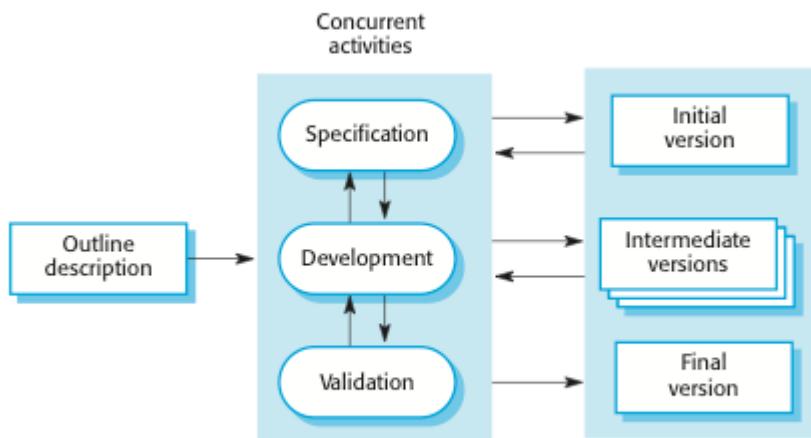
Problémy vodopádového modelu

- Neflexibilné rozdeľovanie projektu do rozličných fáz spôsobuje problémy pri reakcii na meniace sa požiadavky zákazníka.
- Preto je tento model vhodný iba ak sú požiadavky dobre pochopené a zmeny budú pomerne obmedzené počas procesu návrhu.
- Málo biznis systémov má stabilné požiadavky
- Vodopádový model je najčastejšie používaný pre veľké projekty systémového inžinierstva kde je systém vyvíjaný na niekoľkých miestach

Evolučný vývoj

- Prieskumný vývoj

- o Cieľom je spolupracovať so zákazníkmi a vyuvíjať finálny systém z pôvodného návrhu špecifikácie. Mal by začať s dobre porozumenými požiadavkami a pridať nové funkcie ako navrhuje/požaduje zákazník.
- Odhad' – preč prototypovanie :D :D
 - o Cieľom je pochopiť systémové požiadavky. Malo by sa začať so zle pochopenými požiadavkami aby sa určilo čo je naozaj potrebné.

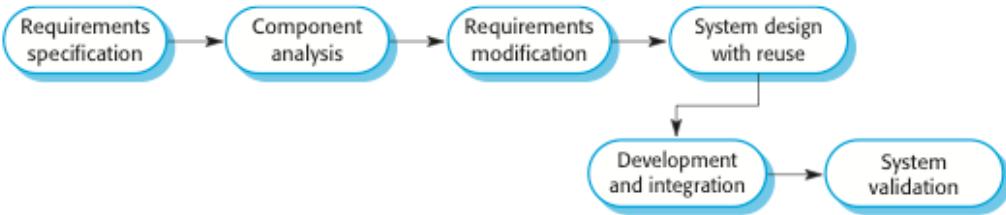


- Problémy
 - o Nedostatok viditeľnosti.
 - o Systémy sú často zle štruktúrované;
 - o Môžu byť požadované špeciálne zručnosti (napr. v jazykoch pre rýchle prototypovanie).
- Použiteľnosť
 - o Pre malé a stredné veľké interaktívne systémy;
 - o Pre časti veľkých systémov (napr. užívateľské rozhranie);
 - o Pre krátkodobé systémy.

Vývoj založený na existujúcich komponentoch

- Založený na systematickom opäťovnom používaní keď sú systémy integrované z existujúcich komponentov alebo COT systémov (Commercial-off-the-shelf).
- Fázy procesu
 - o Analýza komponentov;
 - o Zmeny požiadaviek
 - o Návrh systému so znovupoužívaním;
 - o Vývoj a integrácia.
- Tento prístup je viac a viac používaný odkedy sa objavili štandardy komponentov.

Vývoj orientovaný na znovupoužívanie



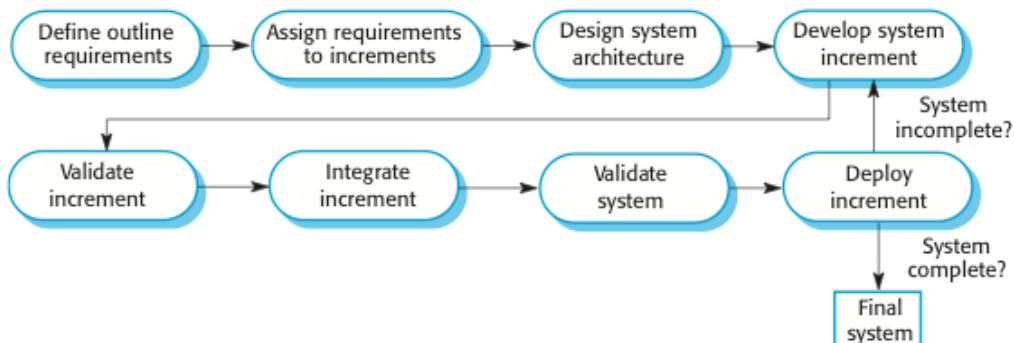
Iterácia procesu

- Systémové požiadavky sa vždy objavia v priebehu projektu tak iterácia procesu, kde skoršie štádiá sú prerobené je vždy súčasťou procesu pre veľké systémy.
- Iterácia môže byť aplikovaná na hociktorý generický model procesu.
- Dva (príbuzné) prístupy
 - o Čiastkové (Inkrementálne) dodávky;
 - o Špirálový vývoj.

Inkrementálne (čiastkové) dodávky

- Skôr ako dodať systém ako celok, vývoj a dodávanie je rozdelené do častí, ktoré dodaní doplnia časť požadovanej funkcionality.
- Požiadavky užívateľov sú prioritné a najvyššie prioritné požiadavky, sú uvedené v prvých častiach.
- Akonáhle je vývoj časti spustený, požiadavky sú zastavené ale požiadavky pre ďalšie časti sa môžu ďalej vyvíjať.

Inkrementálny (čiastkový) vývoj



Výhody čiastkového vývoja

- Hodnota zákazníka môže byť dodaná s každou časťou takže fukcionalita systému je dostupná skôr.
- Skoršie časti fungujú ako prototyp na zistenie požiadaviek pre ďalšie časti.
- Nižšie riziko celkového zlyhania projektu.
- Služby systému s najvyššou prioritou sú najviac testované.

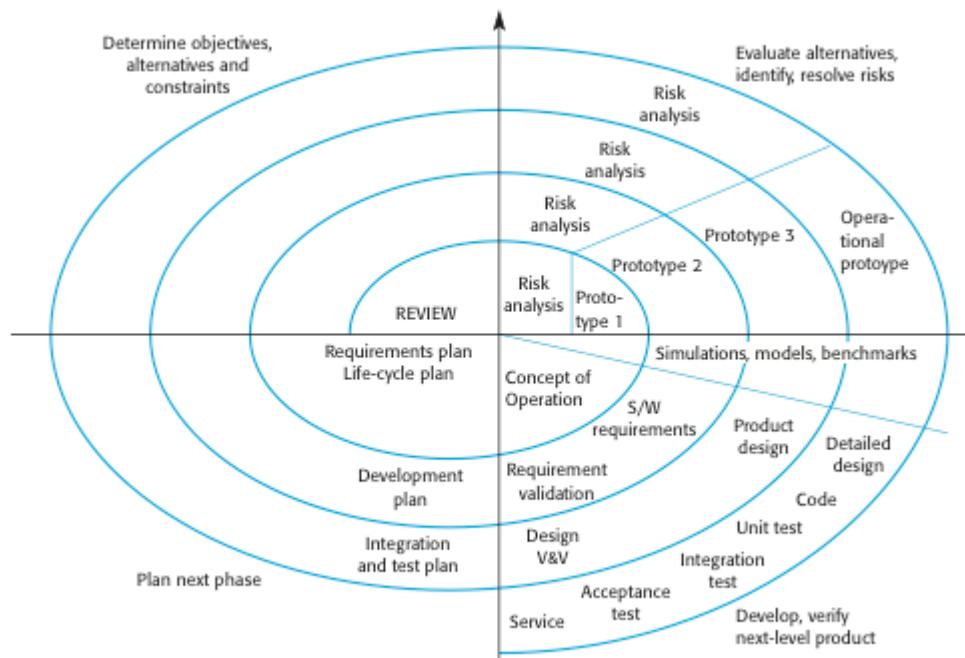
Extrémne programovanie

- Prístup k vývoju založený na vývoji a dodaní veľmi malých častí funkcionality.
- Spolieha sa na konštantné vylepšovanie kódu, zapojenie užívateľov do vývojového tímu a pairwise programovanie.

Špirálny vývoj

- Proces je reprezentovaný ako špirála aktivít nie ako sled aktivít so spätným trasovaním.
- Každá slučka v špirále reprezentuje fazu procesu.
- Nie sú tu žiadne fixné fázy ako špecifikácia alebo dizajn – slučky v špirále sú vyberané podľa toho čo je potrebné.
- Riziká sú explicitne posudzované a riešené po celú dobu procesu.

Špirálny model softvérového procesu



Sektory špirálového modelu

- Stanovenie cieľov

- o Špecifické ciele fáze sú identifikované.
- Posúdenie a zníženie rizík
 - o Riziká sú posudzované a sú zavedené aktivity na zníženie hlavných rizík.
- Vývoj a validácia
 - o Model vývoja pre systém, ktorý je zvolený môže byť jeden zo generických modelov.
- Plánovanie
 - o Projekt je preskúmaný a ďalšia fáza špirály je plánovaná.

Kľúčové body

- Softvérové procesy sú aktivity zapojené pri výrobe a vývoji softvérového systému.
- Modely softvérového procesu sú abstraktné reprezentácie týchto procesov.
- Hlavné aktivity sú špecifikácia, dizajn a implementácia, overenie a evolúcia.
- Generický model procesu opisuje organizáciu softvérových procesov. Príklad zahrňa vodopádový model, evolučný vývoj a vývoj založený na existujúcich komponentoch.
- Iteratívne modely procesu opisujú softvérový proces ako cyklus aktivít.

Softvérové procesy 2

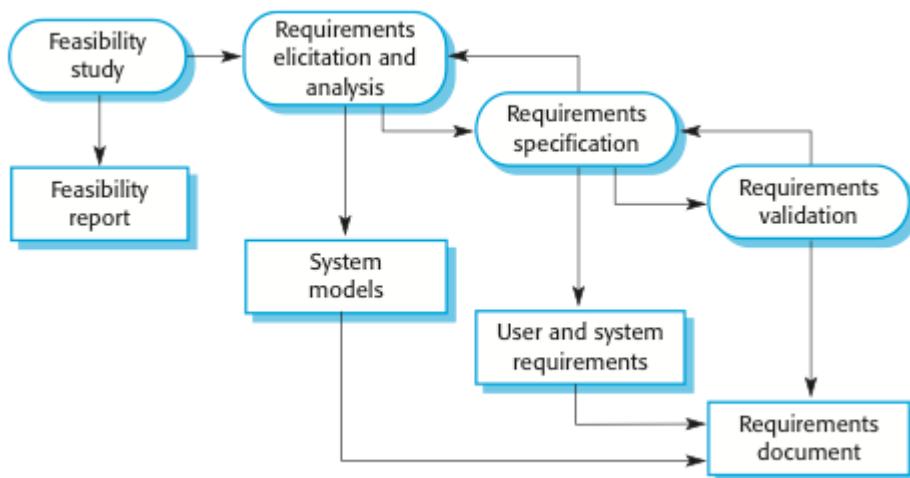
Aktivity procesu

- Špecifikácia softvéru
- Dizajn a implementácia softvéru
- Overenie softvéru
- Evolúcia softvéru

Špecifikácia softvéru

- Proces určenia aké služby a obmedzenia na prevádzku a vývoj systému sú vyžadované
- Requirements engineering process
 - o Štúdia uskutočniteľnosti;
 - o Zisťovanie a analýza požiadaviek;
 - o Špecifikácia požiadaviek;
 - o Validácia požiadaviek.

The requirements engineering process



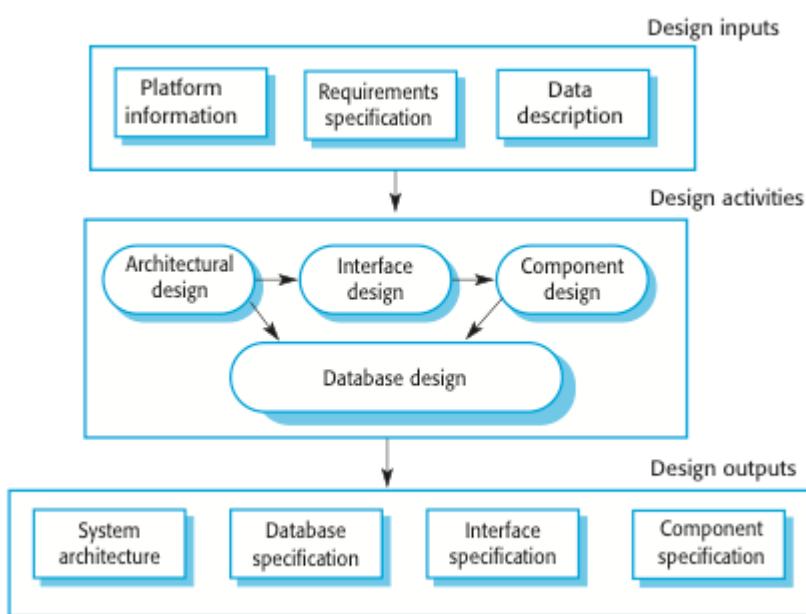
Software design and implementation

- Proces konvertovania systémovej špecifikácie do spustiteľného systému
- Dizajn softvéru
 - o Dizajn a softvérové štruktúry ktoré realizujú špecifikáciu;
- Implementácia
 - o Preloženie tejto štruktúry do spustiteľného programu;
- Aktivity dizajnu a implementácie sú blízko spolu súvisiace a môžu byť popletené

Design process activities

- Architectural design – dizajn vzhľadom na architektúru
- Abstract specification – abstraktná špecifikácia
- Interface design – dizajn rozhrania
- Component design – dizajn komponentov
- Data structure design – dizajn dátových štruktúr
- Algorithm design – dizajn algoritmov

The software design process



Structured methods

- Systematické prístupy k vývoju dizajnu softvéru.
- Dizajn je zvyčajne dokumentovaný ako množina grafických modelov.
- Možné modely
 - o Object model;
 - o Sequence model;
 - o State transition model;
 - o Structural model;
 - o Data-flow model.

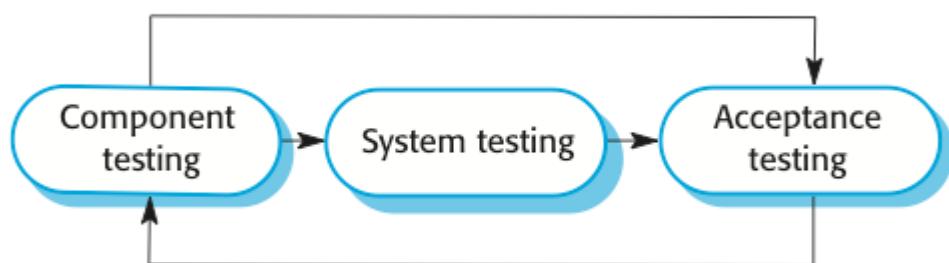
Programming and debugging

- Preklad dizajn do programu a odstránenie chýb z tohto programu.
- Programovanie je osobná aktivita – neexistuje generické programovanie procesu.
- Programátori používajú nejaké testovanie programu na objavenie chýb programu a odstránenie týchto chýb v procese ladenia.

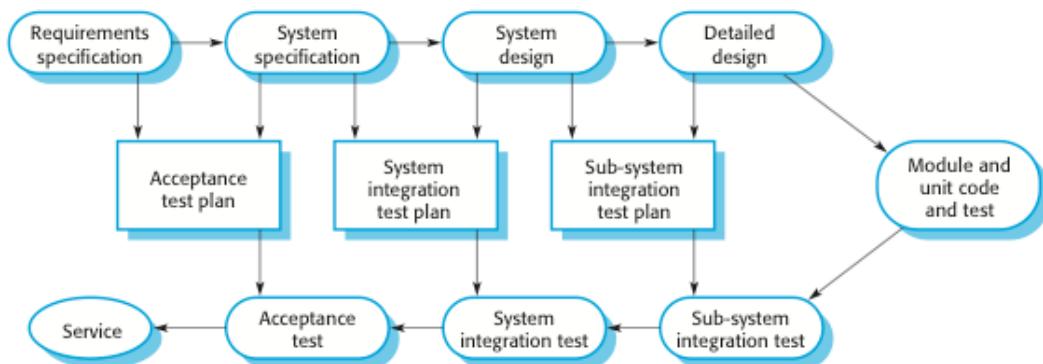
Software validation

- Verifikácia a validácia (V a V) je určená na ukázanie že systém splňa svoju špecifikáciu a spĺňa aj požiadavky systémového zákazníka.
- Zahŕňa kontrolu a revízie procesov a testovanie systému.
- Testovanie systému zahŕňa spúšťanie systému s testovacími prípadmi, ktoré sú odvodené z špecifikacie reálnych dát, ktoré majú byť spracované systémom.

The testing process



Testing phases

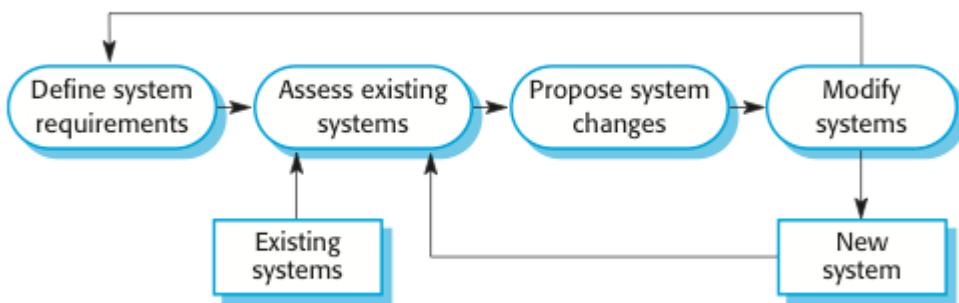


Software evolution

- Softvér je vo svojej podstate pružný a môže sa meniť.
- Ako sa menia požiadavky kvôli zmenám obchodných okolností, softvér ktorý podporuje obchod sa musí vyvíjať a meniť.

- Hoci došlo k vymedzeniu medzi vývojom a evolúciou (údržba), toto je viac a viac irrelevantné lebo menej a menej systémov je úplne nových.

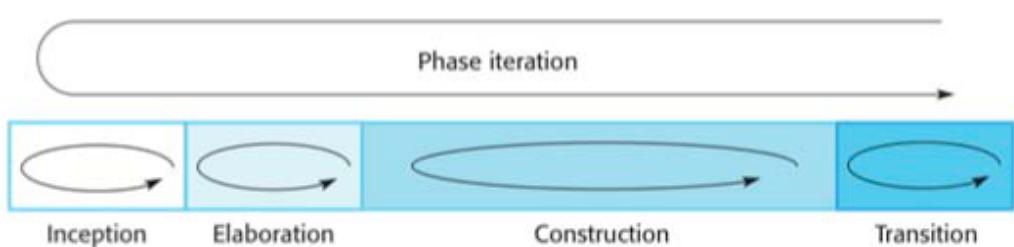
System evolution



The Rational Unified Process

- Moderný procesný model odvodený od pracovania na UML a súvisiaci proces.
- Zvyčajne opisovaný z 3 perspektív
 - o Dynamická perspektíva, ktorá ukazuje fázy v čase;
 - o Statická perspektíva, ktorá ukazuje aktivity procesu;
 - o Praktická perspektíva ktorá navrhuje dobré praktiky

RUP phase model



RUP phases

- Inception – počiatok
 - o Vytvoriť obchodný prípad pre systém
- Elaboration – vypracovanie

- o Rozvíjať porozumenie problémovej domény a systémovej architektúry.
- Construction – stavba
 - o Návrh systému, programovanie a testovanie.
- Transition – modulácia
 - o Nasadenie systému v jeho prevádzkovom prostredí.

RUP good practice

- Vývoj softvéru iteratívne
- Spravovanie požiadaviek
- Použitie component-based architektúry
- Vizuálne modelovať softvér
- Overiť kvalitu softvéru
- Kontrolovať zmeny programu

Static workflows

Workflow	Opis
Business modelling	Obchodné procesy sú modelované požitím obchodných prípadov použitá.
Requirements	Subjekty, ktoré interagujú so systémom sú identifikované a prípady použitia sú vyvinuté pre modelovanie systémových požiadavok.
Analysis and design	Dizajn modelu je vytvorený a zdokumentovaný pomocou architektonických modelov, modelov komponentov, objektových modelov a sekvenčných modelov.
Implementation	Komponenty v systéme sú implementované a štruktúrované do realizácie sub-systémov. Automatické generovanie kódu z modelov návrhov prispieva k urýchleniu tohto procesu.
Test	Testovanie je opakujúci sa proces, ktorý sa vykonáva v súvislosti s implementáciou. Testovanie systému nasleduje po dokončení implementácie.
Deployment	Verzia produktu je vytvorená, distribuovaná používateľom a inštalovaná na pracovisku.
Configuration and	Tento podporný workflow spravuje

change management	zmeny systému.
Project management	Tento podporný workflow riadi vývoj systému.
Environment	Tento pracovný postup sa týka vytvárania vhodných softvérových nástrojov pre vývojový tím softvéru

Computer-aided software engineering

- Computer-aided software engineering (CASE) je softvér pre podporu vývoja softvéru a vývojových procesov.
- Aktivita automatizácie
 - Grafické editory pre vývoj modelu systému;
 - Údajový slovník pre spravovanie konštrukčných prvkov;
 - Grafický UI Builder pre výstavbu používateľského rozhrania;
 - Debuggery na podporu nájdenia chýb;
 - Automatizované prekladače na vytvorenie nových verzii programu.

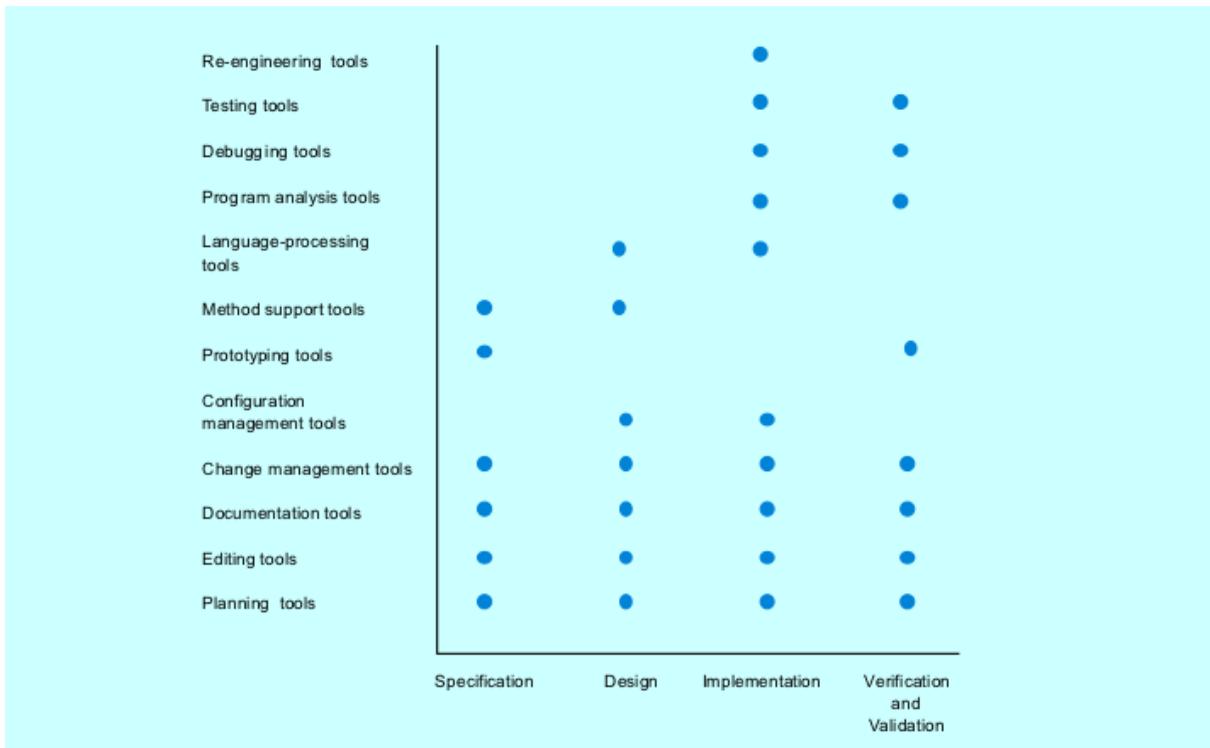
Case technology

- CASE technológia viedla k výraznému zlepšeniu softvérového procesu
However, these are not the order of magnitude improvements that were once predicted (proste to asi neje take dobre jak dufali)
 - Softvérové inžinierstvo vyžaduje kreatívne myšlenie - to nie je jednoducho automatizované;
 - Softvérové inžinierstvo je tímovou činnosťou a, pre veľké projekty, veľa času je stráveného v tímových interakciach. CASE technológie toto nepodporujú.

CASE classification

- Klasifikácia nám pomáha pochopiť rôzne typy CASE nástrojov a ich podporu pre aktivity procesu.
- Funkčná perspektíva
 - Nástroje sú rozdelené podľa ich špecifickej funkcie
- Perspektíva procesu
 - Nástroje sú rozdelené podľa aktivít procesu, ktoré sú podporované.
- Perspektíva integrácie
 - Nástroje sú rozdelené podľa ich organizácie do integrovaných celkov

Activity-based tool classification



Key points

- Požiadavkové inžinierstvo je proces vývoja softvérovej špecifikácie.
- Proces dizajnu a implementácie prevedú špecifikáciu na spustiteľný program.
- Validácia zahŕňa overenie, či systém splňa jeho špecifikácie a potreby užívateľov.
- Evolution sa týka úpravy systému po tom, čo je v prevádzke.
- Rational Unified Process je generický model procesu, ktorý oddeluje činnosti od fáz.
- CASE technológia podporuje aktivity softvérového procesu.

Project management

3prednaška1part

Objectives (cieľe/hlavné body)

- Vysvetliť hlavné úlohy vykonávané projektovými manažérmi
- Predstaviť softvérový projektový manažment a popísať jeho charakteristické vlastnosti
- Diskutovať plánovanie projektu a plánovanie procesu
- Zobraziť, ako sú grafické znázornenia plánov používané v projektovom manažmente

Software project management (softvérový projektový manažmentis)

- Zaoberá sa činnosťami podielajúcimi sa na zabezpečenie toho, že softvér je doručený včas a podľa plánu v súlade s požiadavkami organizácií vyvíjajúcimi softvér
- Projektový manažment(riadenie) je nutný, pretože vývoj softvéru je vždy predmetom rozpočtu a plánovaného obmedzenia, ktoré sú dané organizáciami vyvíjajúcimi softvér

Software management distinctions (úroveň softvérového manažmentu)

- Tento produkt je nehmotný.
- Produkt je jedinečne flexibilný.
- Softvérové inžinierstvo nie je brané(rozpoznávané) ako inžiniersky odbor s rovnakým statusom ako mechanické alebo elektronické inžinierstvo atď
- Proces vývoja softvéru nie je štandardizovaný
- Mnoho softvérových projektov je "jednorazovým" typom projektu

Management activities (činnosti v oblasti riadenia/manažmentu)

- Návrh písania
- Projektové plánovanie a rozvrhovanie
- Výpočet nákladov projektu
- Monitorovanie projektu a hodnotenie
- Personálny/Osobný výber a celkové hodnotenie
- Písanie správ a prezentácií

Project planning (plánovanie projektu)

- Pravdepodobne najviac časovo náročná činnosť počas riadenia (plánovania) projektu
- Neustála činnosť od počiatočného konceptu až po dodanie systému. Ak sú k dispozícii nové informácie, plány musia byť pravidelne aktualizované
- Rôzne typy plánu môžu byť vypracované na podporu hlavného softvérového plánu projektu, ktoré sa týkajú harmonogramu a rozpočtu

Types of project plan (typy plánovania projektu)

- Plán akosti/kvality = Popisuje kvality postupov a noriem, ktoré budú použité v projekte
- Plán overenia = Sú popísané prístupy, zdroje a plány používané na overenie systému
- Plán riadenia konfigurácie = Popisuje postupy riadenia konfigurácie a štruktúry, ktoré majú byť použité
- Plán údržby = Predpovedá požiadavky na údržbu systému, náklady na údržbu a nutné úsilie

The project plan (plán projektu) stanovuje:

- Prostriedky k dispozícii
- Rozpis práce
- Rozvrh práce

Project plan structure(štruktúra plánu projektu)

- Úvod
- Organizácia projektu
- Analýza rizík
- Hardvérové a softvérové požiadavky
- Práca zrútenia (work breakdown)
- Harmonogram projektu
- Monitorovacie a informačné mechanizmy

Activity organization(organizačná činnosť)

- Aktivity v rámci projektu by mali byť organizované tak, aby profukovali hmatateľné výstupy pre ohodnotenie pokroku
- Míľníky sú koncový bod procesnej činnosti
- Výstupy projektov sú výsledky dodávané zákazníkom
- Waterfall proces umožňuje jednoznačnú určenie pokroku míľníkov.

Milestones in the RE process -> vid' obr.

Project scheduling (plánovanie projektu)

- Rozdelenie projektu do úloh, odhad času a zdrojov potrebných na vykonanie jednotlivých úloh
- Usporiadanie úloh súčasne, optimálne využitie pracovnej sily
- Minimalizovať závislostí úloh, aby sa zabránilo oneskoreniu (jedna úloha čaká na dokončenie ďalšej)
- Spol'ahnutie sa na intuíciu a skúsenosť projektových manažérov

The project scheduling process -> vid obr.

Scheduling problems (plánovanie problémov)

- Odhad obtiažnosti problémov a tým aj odhad nákladov na vývoj riešení
- Produktivita nie je úmerná počtu ľudí, ktorí pracujú na úlohe
- Pridanie ľudí do konca projektu, robí to neskôr, pretože komunikačných režijných nákladov
- Neočakávané vždy sa stane. Vždy vedieť pohotovostne plánovať

Bar charts and activity networks (stĺpové grafy a sietové aktivity)

- Grafické notácie používané pre ilustráciu harmonogramu projektu
- Ukázať projekt a rozdeliť do úloh. Úlohy by nemali byť príliš malé. Mali by trvať asi týždeň alebo dva
- Aktivity grafy ukazujú úlohy závislosti a kritickú cestu
- Stĺpcové grafy ukazujú plán na kalendárne odobie

Risk management (riadenie rizík)

- Riadenie rizík sa zaoberá identifikáciou rizík a vypracovania minimalizácie ich vplyvu na projekt
- Riziko = pravdepodobnosť, že niektoré negatívne okolnosti môžu nastať
 - Projektové riziká ohrozujú plán alebo zdroje
 - Produktové riziká majú vplyv na kvalitu alebo výkon softvéru, kt. sa vyvíja
 - Podnikateľské riziká ovplyvňujú organizačný rozvoj alebo obstarávanie softvéru

Software risks -> vid obr.

The risk management process -> vid obr.

Risk indicators -> vid obr.

Key points (klúčové body)

- Dobré riadenie projektu je nevyhnutné pre úspech projektu
- Nehmotná povaha softvéru spôsobuje problémy pre riadenie/správu
- Menežeri sú - majú rôzne role, ale medzi ich najvýznamnejšie aktivity patrí plánovanie, odhadovanie a rozvrhovanie
- Plánovanie a odhadovanie sú opakujúce sa procesy, ktoré pokračujú v celom priebehu projektu
- Etapy projektu sú predvídateľným stavom, kde oficiálne správy o pokroku sú poskytnuté vedeniu
- Projektové plánovanie zahrňa prípravu rôznych grafických znázornení ukazujúcich projektové aktivity, ich trvanie a rozdelenie pracovných funkcií
- Riadenie rizík - sa zaoberá identifikáciou rizík, ktoré môžu mať vplyv na projekt a plánovanie, aby zabezpečilo, že tieto riziká sa nevyvinú v hlavné hrozby

Všetko :D

Software change management

3prednaška2part

Objectives (cieľe/hlavné body)

- Vysvetliť význam softvéru pre správu zmien
- Popísat hlavné činnosti riadenia zmien v oblasti plánovania, riadenia zmien, správu verzií a systémového stavia
- Diskutovať o využití CASE nástrojov pre podporu procesov riadenia zmien

Change management (riadenie zmien)

- Softvérové systémy sú predmetom neustálej zmeny požiadaviek:
 - Od užívateľov
 - Od vývojárov
 - Z trhových síl
- Riadenie zmien sa zaobera sledovaním týchto zmien a zabezpečuje to, aby sa robili v čo najviac cenovo-efektívnym spôsobom

Change management requirements(požiadavky na riadenie zmien)

- Niektoré prostriedky pre používateľov a vývojárov navrhnuté potrebné systémové zmeny
- Proces rozhodovania sa, či by zmeny mali byť zahrnuté do systému
- Softvér sledujúci navrhované zmeny a ich stav
- Softvérová podpora pre riadenie konfigurácie systému a budovanie nových systémov

Change request form (formy žiadosti o zmenu(asi?))

- Definícia formy žiadosti o zmenu je súčasťou CM procesu plánovania
- Táto forma zaznamenáva navrhované zmeny, zmeny žiadateľa, dôvod prečo bola zmena navrhnutá a naliehavosť zmeny (od žiadateľa zmeny)
- To tiež zaznamenáva zmenu hodnotenia, analýzu dopadov, zmenu ceny a odporúčania (systém personálnej údržby)

Change tracking tools (zmena nástrojov pre sledovanie)

- Veľkým problémom v oblasti riadenia zmien je stav sledovania zmien
- Zmena nástrojov pre sledovanie sleduje stav každej žiadosti o zmenu a automaticky zaistí, že žiadosti o zmenu sú odosielané k správnym ľuďom v správny čas
- Integrácia s e-mail systémami umožňujúcimi elektronické žiadosti o zmenu distribúcie

Change approval (schválenie zmeny)

- Zmeny by mali byť preskúmané externou skupinou, ktorá rozhodne, či sú alebo nie sú cenovo efektívne zo strategického a organizačného hľadiska, skôr než technického hľadiska
- Schválenie zmeny by malo byť nezávislý na projekte zodpovedným za systém. Táto skupina je niekedy nazývaná zmena ovládacieho panelu
- CCB môže zahŕňať zástupcov klienta a zamestnancov od dodávateľa

Derivation history (história odvodenia?)

- Je to záznam zmien aplikovaných na dokumente alebo časti kódu
- História by mala zaznamenať, v obryse, vykonanú zmenu, dôvody pre zmenu, kto zmenu vykonal a kedy bola vykonávaná
- Môže byť zahrnutá ako komentár v kóde. Ak sa používa štandardný prológ štýl na odvodenie histórii, nástroje ich môžu spracovať automaticky

Configuration management(riadenie konfigurácie)

- Nové verzie softvérových systémov sú vytvorené počas zmien:
 - Pre rôzne stroje / OS
 - Ponúka rôzne funkcie
 - Na mieru pre konkrétné užívateľské požiadavky
- Riadenie konfigurácií sa zaoberá riadením vyvíjajúcich sa softvérových systémov:
 - Zmena systému je tímová činnosť
 - CM si kladie za cieľ kontrolovať náklady a úsilie spojené s vykonaním zmien v softvérovom systéme a súvisiacej dokumentácie
- Zahŕňa vývoj a uplatňovanie postupov a noriem vyvíjajúcich softvérový produkt
- CM môže byť súčasťou všeobecnejšieho procesu riadenia kvality
- Po uvoľnení CM, softvérové systémy sú niekedy nazývaná baselines(základné línie) a sú východiskom pre ďalší vývoj

The configuration management plan(plán riadenia konfigurácie)

- Definuje typy dokumentov, ktoré majú byť riadené a schematické pomenovanie dokumentov
- Definuje, kto preberá zodpovednosť za CM postupy a vytváranie línií
- Definuje stratégiu pre riadenie zmeny a správu/riadenie verzií

- Definuje CM záznamy ktoré musia byť zachované
- Popisuje nástroje ktoré by mali byť použité na pomoc CM procesu a prípadné obmedzenie ich používania
- Definuje proces použitia nástrojov
- Definuje CM databázu použitej pre záznam informácií o konfigurácii
- Môže obsahovať informácie, ako je CM externého softvéru, procesné audity, atď.

Release management (vydanie riadenia?)

- Správy/Vydania musia obsahovať zmeny vynútené systémom, chyby zistené užívateľmi a zmeny hardvéru
- Musia tiež obsahovať novú funkčnosť systému
- Release plánovanie sa zaoberá kedy vydať novú verziu systému

System releases (systémové správy)

- Nielen to len sada spustiteľných programov.
- Môže tiež obsahovať:
 - Konfiguračné súbory definujúce, ako je systém nakonfigurovaný pre konkrétnu inštaláciu
 - Dátové súbory potrebné na prevádzku systému
 - Inštalačný program alebo skript(shell) pre inštaláciu systému na cieľovom hardvéri
 - Elektronická a tlačená dokumentácia
 - Balenie a súvisiace reklamy
- Systémy sú teraz bežne vydávané na optické disky (CD alebo DVD) alebo ako stiahnuteľný obsah na prevzatie inštalačných súborov z webu

System building (budovanie systému)

- Proces komplikácie a prepojenia softvérových komponentov do spustiteľného systému
- Rôzne systémy sú budované z rôznych kombinácií komponentov
- Tento proces je teraz vždy podporovaný automatizovanými nástrojmi, ktoré sú poháňané by "build skripts"

- Môžu to byť samostatné nástroje (napr. make) alebo časť prostredia programovacieho jazyka (rovnako ako v Jave)

System building problems(problémy v budovaní systému)

- Majú budovacie inštrukcie všetky požadované súčasti?
 - Ked' tam sú stovky komponentov, ktoré tvoria systém, je ľahké jeden prehliadnuť .To by malo byť zvyčajne detekované linkermi
- Je uvedená vhodná verzia komponentu?
 - Výraznejší problém. Systém postavený na nesprávnej verzii môže spočiatku fungovať, ale potom spadne po doručení
- Sú všetky dátové súbory k dispozícii?
 - Build by sa nemal spoliehať na "štandardné" dátové súbory. Niekedy sa normy líšia
- Sú dátové odkazy na súbory správne?
 - Vloženie absolútnych mien v kóde takmer vždy spôsobuje problémy. Spôsoby pomenovania sa líšia
- Systém je postavený na správnej platforme?
 - Niekedy je nutné vytvoriť systém pre konkrétnu verziu operačného systému alebo konfiguráciu hardvéru
- Je správne verzie kompilátora a iných softvérových nástrojov určená?
 - Rôzne verzie kompilátora môžu v skutočnosti generovať iný kód a výsledok bude vykazovať odlišné správanie

Change management tools(zmena nástrojov pre správu/riadenie)

- CM postupy sú štandardizované a zahrňajú použitie vopred definovaných postupov
- Veľké množstvo dát musí byť riadené
- Pomocné nástroje CM sú preto zásadné
- Staršie CASE nástroje na podporu riadenia konfigurácie sú k dispozícii v rozsahu od samostatných nástrojov až po integrované CM stoly(?)

The configuration database (konfigurácia databázy)

- Všetky CM informácie by mali byť udržiavané v konfiguračnej databáze.
- To môže vytvárať otázky ohľadom konfigurácie:
 - Kto má konkrétnu verziu systému?
 - Aká platforma je nutné pre konkrétnu verziu?
 - Aké verzie sú ovplyvnené zmenou súčiastky X?
 - Koľko hlásených chýb je vo verzii T?
- CM databázy by mali byť prednostne spojené so spravovaným softvérom

Change control tools (zmena kontrolných nástrojov)

- Zmena kontroly je procesný postup, takže môže modelovať a integrovať so systémom pre správu/riadenie verzií
- Zmena kontrolných nástrojov
 - Forma editoru na podporu spracovania zmenu-žiadajúcich formulárov
 - Workflow systém definovať, kto čo robí, a automatizovať prenos informácií
 - Zmeniť databázu ktorá spravuje pozmeňovacie návrhy a je spojená so systémom VM
 - Zmeniť systém podávania správ, ktorý generuje riadenie správ o stave požiadaviek na zmeny

Version management tools (verzie riadiacich nástrojov)

- Verzia a identifikácia
 - Systémy priradia identifikátory automaticky k novej verzii priloženej k systému
- Správa úložiska
 - Systém ukladá rozdiely medzi verziami ale nie na všetky verzie kódu
- Záznam histórií zmien
 - Zaznamenáva dôvody tvorby verzie
- Nezávislý vývoj

- Iba práve jedna verzia môže byť vydaná na zmenu. Paralelné pracovanie na rôznych verziách
- Podpora projektu
 - Môže spravovať skupiny súborov súvisiacich s projektom, skôr ako len jednotlivé súbory

System building(budovanie systému)

- Budovanie veľkých systémov je výpočtovo náročné a môže trvať niekoľko hodín
- Stovka súborov môže byť zapojená
- Budovanie systému môže poskytovať
 - Závislosti špecifikácie jazyka a interpretéra
 - Nástroj pre výber a konkretizáciu podpory
 - Distribuovanú komplikáciu
 - Odvodené objekty riadenia

Key points (klúčové body)

- Riadenie zmien sa zaoberá všetkými činnosťami okolo navrhovanie, posudzovanie a vykonávanie zmien v softvérovom systéme
- Zmena riadenia zahŕňa posúdenie technickej a ekonomickej životoschopnosti zmien návrhov
- Konfiguračný manažment je proces riadenia rozvíjajúci softvérový produkt
- Zmena nástrojov pre správu môžu byť samostatné nástroje alebo to môžu byť integrované systémy, ktoré integrujú podporu pre riadenie zmien, správu verzií, budovania systému a riadenia zmien

Softvérové požiadavky

- proces pri ktorom sa ustanovujú zákaznícke požiadavky na systém a obmedzenia systému na ktorom sa software vytvára a pracuje.
- požiadavky sami o sebe sú opisom systémových služieb a obmedzení ktoré sa vyskytnú počas tohto procesu.

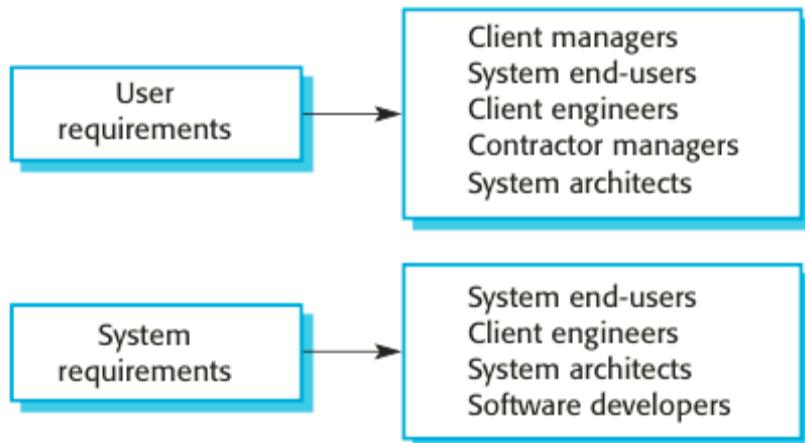
Požiadavka : rozsah požiadaviek môže ísť od vysokej miery abstraktnosti servisového príkazu alebo systémového obmedzenia až po detailnú matematickú funkcionálnu špecifikáciu.

Požiadavky môžu slúžiť ako duálne funkcie(vraj je to nevyhnutné)

- Môže byť základom pre ponuku kontraktu : preto musí byť otvorená pre výklad/interpretáciu.
- Môže byť základom pre samotný kontrakt : preto musí byť detailne definovaná.
- Oba výroky môžu byť nazvané požiadavkami.

Typy požiadaviek :

- Používateľské požiadavky
 - Sú písane v prirodzenom jazyku + diagramy služieb ktoré poskytuje systém a ich operačné obmedzenia.
- Systémové požiadavky
 - Štruktúrovaný dokument popisujúci detailný opis systémových funkcií, služieb a operačných obmedzení. Definuje čo má byť implementované, takže môže byť časť kontraktu medzi klientom a dodávateľom.



Funkcionálne požiadavky

- Služby, ktoré by mal systém poskytovať, ako má systém reagovať na jednotlivé vstupy a ako sa má systém správať v jednotlivých situáciach.
- Popisujú funkciaľitu alebo systémové služby.
- Závisia na type softwaru, predpokladajú používateľov a typ systému kde je softvér používaný.

- Funkcionálne používateľské požiadavky môžu byť všeobecné vyjadrenia čo by mal systém robiť ale funkcionálne systémové požiadavky by mali popisovať systémové služby detailne.

Nefunkcionálne požiadavky

- Obmedzenie služieb alebo funkcií ponuknutých systémom ako časové obmedzenia, obmedzenia vývojového procesu, štandardy, atď.

Doménové požiadavky

- Požiadavky ktoré prídu z domény systémových aplikácií a prejavia sa v charakteristikách domény.

LIBSYS systém

- Knižničný systém ktorý poskytuje jedno rozhranie na prepojenie k množstvu databázových článkov v rôznych knižničiach.
- Používateľ prehľadávať, stahovať a tlačiť tieto články pre osobné štúdium.

Príklady funkcionálnych požiadaviek

- Používateľ by mal byť schopný budť celú databázu alebo iba jej podčasti.
- Systém by mal poskytovať vhodných wiewers pre užívateľa na čítanie dokumentov z dokumentového skladu.
- Každá objednávka by mala mať alokovaný jedinečný identifikátor(id) ktorý by mal byť používateľ schopný skopírovať do úložného priestoru svojho účtu.

Nepresnosti požiadaviek

- Problémy vznikajú keď požiadavky niesú presne stanovené.
- Viacvýznamové požiadavky môžu byť interpretované odlišne vývojárom a používateľom.
- Zvážte termín „Vhodný zobrazovač“(Appropriate wiewers)
 - Zámer používateľa - špeciálny účel zobrazovača pre každý typ dokumentu
 - Zámer vývojára - poskytnúť textový zobrazovač ktorý zobrazí obsah dokumentu.

Požiadavky : kompletnosť a konzistencia

- V princípe by požiadavky mali byť kompletné a konzistentné.
- Kompletné
 - Mali by obsahovať popis všetkých potrebných vlastností
- Konzistentné
 - Nemali by obsahovať žiadny konflikt alebo protirečenie v popise systémových vlastností.
- V praxi, je nemožné písat kompletné a konzistentné požiadavky.

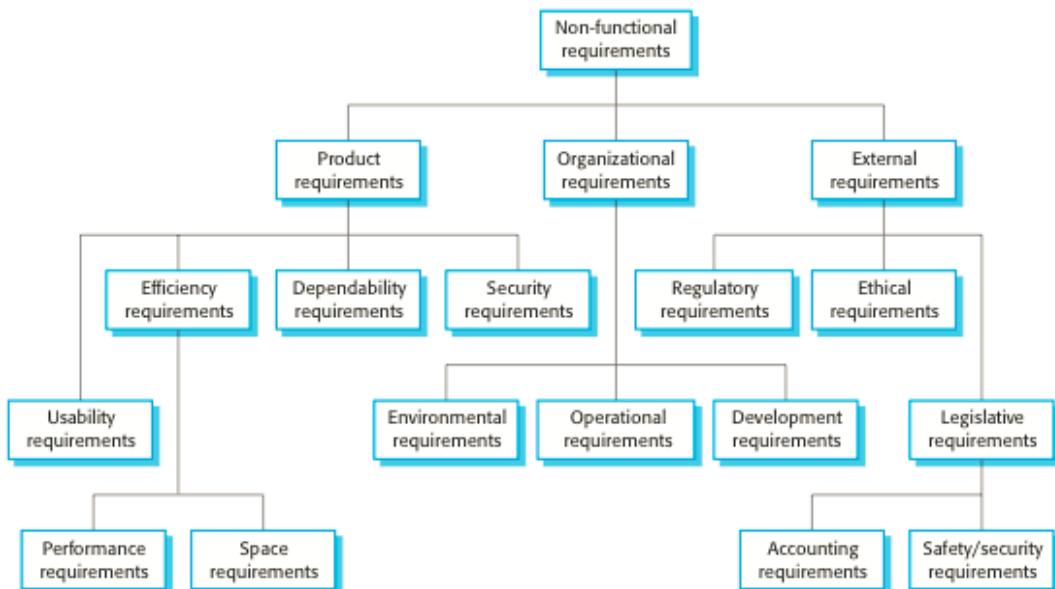
Nefunkcionálne požiadavky

- Definujú systémové vlastnosti a obmedzenia ako spoľahlivosť, čas odozvy a pamäťové požiadavky. Obmedzenia sú I/O schopnosti zariadení, reprezentácia systému, atď.

- Proces požiadaviek môže byť špecifikovaný povereniami jednotlivých CASE systémov, programovacími jazykmi alebo vývojovými metódami.
- Nefunkcionálne požiadavky môžu byť kritickejšie ako funkcionálne požiadavky. Ak niesú splnené systém je nepotrebný(useless).

Nefunkcionálne pož. Klasifikácia

- Produktové požiadavky
 - Požiadavky ktoré, špecifikujú, že dodaný produkt sa musí určitým spôsobom správať napr. spúštacia schopnosť, spoľahlivosť, atď.
- Organizačné požiadavky
 - Požiadavky ktoré sú výsledkom organizačných pravidiel a procedúr ako napr. process standards used, implementation requirements, etc.
- Externé požiadavky
 - Požiadavky ktoré vyplývajú z vonkajších faktorov pre systém a jeho vývojový proces ako napr. legislatívne požiadavky, a pod.



Príklady nefunkcionálnych požiadaviek

- Produktové požiadavky
 - o 8.1 používateľské rozhranie pre LIBSYS by malo byť implementované ako jednoduché HTML bez framov alebo Java appletov.
- Organizačné požiadavky
 - o 9.3.2 Vývojový proces systému a doručiteľné dokumenty by sa mali prispôsobiť procesom a doručiteľnosti definovaných v XYZ=SP-STAN-95
- Externé požiadavky

- o 7.6.5 Systém by nemal prezradiť žiadne osobné informácie o zákazníkoch okrem ich mena a ref. čísla pre operátorov systému.

Ciele a požiadavky

- Nefunkcionálne požiadavky môžu byť veľmi zložité na presný popis a nepresné požiadavky môžu byť zložité na overenie.
- Ciele
 - o Všeobecný úmysel používateľov na jednoduché používanie
- Dokázaťnosť nefunkčných požiadaviek
 - o Výraz používa niektoré miery ktoré sa dajú objektívne testovať.
- Ciele sú nápomocné pre vývojárov ako previesť zámery užívateľov
- Goals are helpful to developers as they convey the intentions of the system users.

Príklady

- Systémové ciele
 - o Systém by mal byť ľahko používateľný pre skúsených kontrolorov and mal by byť organizovaný tak aby používateľské chyby boli minimalizované.
- Dokázaťnosť nefunkcionálnych požiadaviek
 - o Skúsený kontrolóri by mali byť schopný používať všetky systémové funkcie po dvoch hodinách tréningu. Po tomto tréningu, priemerný počet chýb vytvorených skúsenými používateľmi by nemal presiahnuť dve chyby za deň.

Property	Measure
Speed	Processed transactions/second User/event response time Screen refresh time
Size	Mbytes Number of ROM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

Interakcia požiadaviek

- Konflikt medzi rôznymi nefunkcionálnymi požiadavkami sú časte v komplexných systémoch.
- Systém vesmírnej lode

- o Aby sme zminimalizovali váhu, počet samostatných čipov v systéme by mal byť minimalizovaný.
- o Znižiť spotrebu energie , treba použiť čipy s nižšou spotrebou energie.
- o Aj keď použijeme nízko-spotrebové čipy môže to znamenať, že ich bude treba použiť viac. Ktorá je najkritickejšia požiadavka?

Kľúčové pojmy

Requirements set out what the system should do and define constraints on its operation and implementation.

□ Functional requirements set out services the system should provide.

□ Non-functional requirements constrain the system being developed or the development process.

□ User requirements are high-level statements of what the system should do. User requirements should be written using natural language, tables and diagrams.

Softverové požiadavky 2

Doménové požiadavky

- Odvodené od aplikačných domén, popisujú systémové charakteristiky a rysy ktoré odrážajú doménu.
- Doménové požiadavky sú novými fukcionálnymi požiadavkami, obmedzujú existujúce požiadavky alebo definujú špecifický postup výpočtov.
- Ak doménové požiadavky niesú splnené, systém sa môže stať nefunkčným.

Knižničný systém doménové požiadavky

- Mal by mať štandardný user interface pre všetky databázy ktoré sú založene na Z39.50 standarde.
- Kôli copyright obmedzeniam, niektoré dokumenty musia byt zmazané hned po príchode.
Záleží to na požiadavkách užívateľa, tieto dokumenty budú buď tlačené lokálne na systémovom serveri pre manuálnu dopravu k používateľovi alebo odoslane na sieťovú tlaciareň.

Vlakový ochranný systém

- Deklarácia vlaku by mala byť vypočítaná ako
 - o Dtrain = Dcontrol + Dgradient
 - o kde Dgradient je $9,81 \text{ ms}^2 * \text{kompenzované gradien}/\alpha$ kde hodnoty $9,81/\alpha$ sú známe pre rôzne typy vlakov

Doménové požiadavky – problémy

- Pochopiteľnosť
 - o Požiadavky sú vyjadrené v jazyku aplikačnej domény.
 - o Toto je často nepochopené softvérovými inžiniermi ktorí vytvárajú systém.
- Implicitnosť
 - o Doménový špecialisti rozumejú tejto oblasti natol'ko, že nevytvárajú explicitné doménové požiadavky.

Užívateľské požiadavky

- Mali by popisovať funkcionálne a nefunkcionálne požiadavky tak aby boli pochopiteľné pre používateľa systému ktorý nemá detailné znalosti.
- Používateľské požiadavky sú definované použitím prirodzeného jazyka, tabuľiek a diagramov aby boli pochopiteľné pre každého používateľa.

Problémy s prirodzeným jazykom

- Nedostatok jasnosti(clarity)
 - o Na dosiahnutie presnosti musí byť všetko dopodrobna popísané ergo v konečnom dôsledku je dokument tažko čitateľný.(aspon dajak tak som pochopil)
- Zmätok v požiadavkách
 - o funkcionálne a nefunkcionálne požiadavky zvyknú byť pomiešané.
- Zlúčenie požiadaviek
 - o Niekoľko rôznych požiadaviek môže byť vyjadrených v jednom.

Pravidlá na písanie požiadaviek

- Vytvoriť štandardný formát a použiť ho pre všetky požiadavky.
- Používať jazyk konzistentne.
- Používať vysvecovanie textu na identifikáciu kľúčových častí požiadaviek.
- Vyhnut sa používaniu pc žargónu.

Systémové požiadavky

- Detailnejšia špecifikácia systémových funkcií, služieb a obmedzení ako používateľských požiadaviek.
- Sú považované za základ pre design systému.
- Môžu byť združené do systémovej dohody.
- Systémové požiadavky môžu byť definované alebo ilustrované pomocou systémových modelov spomínaných v CH8.

Požiadavky a dizajn

- V princípe, požiadavky by mali stanoviť čo by systém mal robiť a dizajn by mal popisovať ako to robí.
- V princípe, požiadavky a dizajn sú neodlúčiteľné
 - Architektúra systému môže byť dizajnovaná na vytvorenie požiadaviek.
 - Systém môže vnútorme pracovať s iným systémom, ktorý generuje dizajnové požiadavky.
 - Na použitie špecifického dizajnu môže byť doménová požiadavka.

Štruktúrovaný jazyk špecifikácií

- Sloboda tvorca požiadaviek je limitovaná preddefinovanými vzormi pre požiadavky.
- Všetky požiadavky sú písané štandardne.
- Terminológia použitá v popise môže byť limitovaná.
- Výhodou je, že väčšina výrazov jazyka je zachovaná ale stupeň rovnosti je predpísaný v špecifikácii.

Form-based špecifikácie

- Definícia funkcie alebo entity.
- Popis vstupov a popis ich pôvodu.
- Popis výstupov a popis kam majú ísť.
- Indikácia iných potrebných entít.
- Predpoklady a (pre/post condition)ak sú potrebné
- Side efekty (ak nejaké sú) funkcií.

Sekvenčné diagramy

- Zobrazujú sekvencie udalostí ktoré prebiehajú v systéme počas komunikácie používateľa so systémom.
- Čítajú sa od vrchu na spodok aby sme videli všetky interakcie systému.
- Výber peňazí
 - skontrolovať kartu

- o požiadavka na vydanie
- o skompletovanie transakcie

Požiadavky na dokumenty(The requirement documents)

- Oficiálne oznamenie čo je požadované od systémových tvorcov.
- Malo by zahrňovať definície používateľských požiadaviek a špecifikáciu systémových požiadaviek.
- Nie je to dizajnový dokument. Pokiaľ je to možné, mal by nastavovať hlavne Čo má systém robiť ako Ako to má robiť.

IEEE požiadavkový štandard

- Definuje spoločnú štruktúru pre požiadavkový dokument ktorý musí byť inštančný pre každý špecifický systém.
 - o Úvod
 - o Všeobecný opis
 - o Špecifické požiadavky
 - o Appendices(slepé črevá) :D
 - o Indexy

Požiadavky na dokumentovú štruktúru

- Predslov
- Úvod
- Slovník/Register
- Definície užívateľských požiadaviek
- Architektúra systému
- Špecifikácia systémových požiadaviek
- Systémové modely
- Evolúcia systému
- Appendices
- Indexy

Key points

- Systémové požiadavky sú určené na komunikáciu funkcií ktoré ma systém ponúkať.
- A software requirements document is an agreed statement of the system requirements.
- IEEE štandard je užitočný štartovací bod na definovanie detailnejších štandardov na špecifikáciu požiadaviek.

System modeling 1 (prednáška 5-1)

Objectives:

- Vysvetliť, prečo by v kontexte systému mala byť modelovaná ako súčasť RE procesu
- K popisu správania modelovania, dátové modelovanie a objektové modelovanie
- Pre zavedenie niektorých zápisov používaných v Unified Modeling Language (UML)
- Ak chcete zobraziť, ako podporiť CASE stoly modelovanie systémov

System modeling:

Modelovanie systému pomáha analytikovi pochopiť funkčnosť systému a modely sú používané pre komunikáciu so zákazníkom.

- Rôzne modely predstavujú systém z rôznych pohľadov
- Vonkajší pohľad zobrazujúci systémový kontext alebo životného prostredia
- „Správajúci sa“ pohľad ukazujúci správanie systému
- Štrukturálny pohľad zobrazujúci systémovú alebo dátovú architektúru.

Model types:

- Spracovanie dát modelu ukazuje, ako sú dátá spracované v rôznych fázach.
- Zloženie modelu ukazuje, ako sú entity zložené iných subjektov.
- Architektonický model, znázorňujúci hlavné subsystémy.
- Klasifikačný model, ktorý ukazuje, ako subjekty majú spoločné črty.
- Stimulus / odozva modelu ukazuje systémovú reakciu na udalosti.

Context models (kontextove modely)

- Kontextové modely sú použité pre ilustráciu prevádzkových podmienok systému, ktoré ukazujú, čo leží mimo hranice systému.
- Sociálne a organizačné otázky môžu mať vplyv na rozhodnutie o tom, kam umiestniť hranice systému.
- Architektonické modely ukazujú systému a jeho vzťahu s ostatnými systémami.

Process models

- Procesné modely ukazujú celkový proces a procesy, ktoré sú podporované systémom.
- Dátový tok modelov môže byť použitý na zobrazenie procesov a toku informácií z jedného procesu na iný.
- Tieto typy diagramu sú niekedy označované ako workflow diagramy

Behavioural models

- Behaviorálne modely sa používajú na opis celkového správania systému.
- Dva typy správania modelu sú:
 - Spracovanie dát, modely, ktoré ukazujú, ako sú dátá spracované, pretože sa pohybuje v systéme;
 - Stavové modely, ktoré ukazujú systémy reakcie na udalosti.
- Tieto modely ukazujú rôzne pohľady takže oba musia opísť správanie systému.

Data-processing models

- (Data flow diagrams) diagramy dátového toku (DFDs) môžu byť použité na modelovanie systému spracovania dát.
- Tie znázorňujú kroky spracovania ako tok dát pomocou systému.
- DFDS sú neoddeliteľnou súčasťou mnohých metód analýzy.
- Jednoduchý a intuitívny zápis, aby zákazníci mohli rozumieť.
- Zobraziť end-to-end spracovanie dát.

Data flow diagrams

- DFDS modelu systému z funkčného hľadiska.
- Sledovanie a zaznamenávanie ako sú údaje spojené s procesom je užitočné na vypracovanie a celkové pochopenie systému.
- Dátové diagramy môžu byť tiež použité v tom, výmenu dát medzi systémom a inými systémami v jeho prostredí.

UML diagrams

- K modelu pracovného postupu alebo toku dát, môžete použiť UML diagramy aktivít 2.
- Tieto poskytujú bohatší notáciu, preklad je jednoduchý
- UML stavový diagram môže byť použitý na model, ako sú spracované udalosti systémom

State machine models

- Tieto modelové správanie systému, sú v závislosti na vonkajších a vnútorných udalostí
- Ukazujú reakcie systému na podnety, takže sa často používajú pre modelovanie v reálnom čase
- State machine models ukazujú stavy systému ako uzly a udalosti sú oblúky medzi týmito uzlami. Keď dôjde k udalosti, systém prechádza z jedného stavu do druhého.
- Statecharts sú neoddeliteľnou súčasťou UML a sú používané na reprezentovanie state machine models

Statecharts

- Dovoľuje rozklad modelu do čiastkových modelov
- Stručný opis aktivít je obsiahnutý v nadväznosti na „rob“ v každom stave
- Môže byť doplnená tabuľkami popisujúcimi stavy a podnety.

The Unified Modeling Language (UML)

- UML sa stal štandardom schematického zápisu pre vytváranie modelov objektovo orientovaných systémov.
- UML je zjednotenie celej rady rôznych prístupov k modelovaniu OO, ktoré boli vyvinuté v roku 1990.
- Jeho najnovšia verzia UML (2) zahŕňa aj všeobecnejšie modelovacie konštrukcie
- UML 1 bol najpoužívanejšie, keď bola napísaná táto kapitola. Preto som použil tento zápis tu. V dôsledku toho nie sú niektoré diagramy ako data-flow vyjadrené v UML.
- Unified (Booch, Rumbaugh, Jacobson) Modelovanie (vizuálne, grafické) Jazyk (syntax, sémantika)
- Object Management Group (OMG) definícia:
- Unified Modeling Language (UML) je grafický jazyk pre vizualizáciu, špecifikovanie, výstavbu a dokumentovanie artefaktov softvérového systému (software-intensive system). UML ponúka štandardný spôsob, ako napísať systém na plány, vrátane koncepcných vecí, ako je obchodný proces a systémová funkcia, ako aj konkrétné veci, ako je programovací jazyk, vyhlásenie, databázových schém a opakovane použiteľných softvérových komponentov.

UML2

- UML 2 má 13 rôznych typov zapojení, ktoré sa používajú na modelovanie rôznych aspektov systému
- Behaviorálne diagramy sa používajú na modelovanie správania funkcie systému. Napríklad, use-case diagramy, diagramy aktivít, stavový stroj diagramy
- Interakčné diagramy sa používajú na modelovanie interakcií medzi jednotkami v systéme. Napríklad, sekvenčné diagramy a diagramy komunikácie.
- Štruktúrne diagramy sa používajú pre modelovanie organizácie systému. Napríklad diagramy tried, diagramy, balenie a vývojové diagramy.

Key points

- Model je abstraktný system view.
- Doplňkové typy modelu poskytujú rôzne informácie o systéme.
- Kontextové modely ukazujú pozíciu systému v jeho prostredí s inými systémami a procesmi.
- Dátový tok modely môžu byť použité na modelovanie spracovania dát v systéme.
- stavové modely modelujú správanie systému v reakcii na vnútorné alebo vonkajšie udalosti

System modeling 2

Semantic data models

- Používa sa na opis logickej štruktúry údajov spracúvaných v rámci systému.
- Subjekt-vzťah-atribút modelu stanovuje subjekty v systéme, vzťahy medzi týmito subjektmi a vzťahy medzi subjektmi
- Široko používaný v návrhu databázy. Môžu byť ľahko implementované pomocou relačnej databázy.
- V UML, objekty a združenia sa môžu použiť pre sémantické modelovanie dát.

Data dictionaries

- Dátové slovníky sú zoznamy všetkých používaných mien v systémových modelov. Popisy vzťahov entít a atribútov sú tiež zahrnuté.
- Výhody
 - Podpora name management a vyhnúť sa duplike
 - Skladuje organizačné znalosti spájajúcej analýzu, návrh a implementáciu
- CASE nástroje niekedy môžu automaticky generovať dátové slovníky od systémových modelov

Object models

- Objekt modely popisujú systém, pokiaľ ide o triedy objektov a ich združenia.
- Trieda objektu je abstrakcia nad súborom objektov so spoločnými atribútmi a služby (operácie) poskytovaných každým objektom
- Rôzne modely objektov môžu byť vyrobené:
 - Dedičné modely
 - Agregačné modely
 - Interakčné modely
- Prírodzené spôsoby ako odrážať entity reálneho sveta manipulované v systémom
- Viac abstraktné entity sú ľahšie modelovať pomocou tohto prístupu
- Trieda objektu identifikácie je uznávaná ako zložitý proces vyžadujúci hlboké porozumenie aplikačnej domény
- Triedy objektov odrážajúcich doménové entity sú opakovane použiteľné naprieč systémami

Inheritance models (dedičné modely)

- Organizujú triedy objektu domény do hierarchie.
- Triedy na vrchole hierarchie odrážajú spoločné črty všetkých tried.
- Triedy objektov dedia atribúty a služby z jedného alebo viacerých super-tried. To môže byť špecializované ak je to nevyhnutné.
- Class hierarchy dizajn môže byť zložitý proces, ak je potrebné sa vyhnúť opakovaniu v rôznych odvetviach

UML notation

- Triedy objektov sú obdĺžniky s názvom v hornej časti, atribúty v strednej časti a operácie v spodnej časti.
- Vzťahy medzi triedami objektov (známy ako združenie) sú zobrazené ako čiary spájajúce objekty. Asociácia môže byť poznámku s typom združenia alebo špeciálne symboly šípkov možno použiť na označenie typu združenia (napr. dedičnosť je trojuholníková biele šípky)
- Dedičnosť je referovaná ako zovšeobecnenie a zobrazuje sa "hore" a nie "dole" v hierarchii.

Multiple inheritance (Viacnásobná dedičnosť)

- Skôr než dediť atribúty a služby z jednej nadradenej triedy, systém, ktorý podporuje viacnásobnú dedičnosť umožňuje objektu triedy dediť z niekoľkých Super-tried
- To môže viesť k sémantickým konfliktom, kde atribúty/služby s rovnakým názvom v rôznych super-triedy majú rôzne sémantiku
- Viacnásobná dedičnosť robí reorganizáciu hierarchických tried zložitejšiu.

Object aggregation

- Model agregácie ukazuje ako sú triedy, ktoré sú kolekcie zložené z iných tried.
- Agregačné modely sú podobné čiastočným vzťahom v sémantických dátových modeloch

Object behaviour modeling (model správanie objektu)

- Model správania ukazuje interakcie medzi objektmi na produkovanie nejakých zvláštnych správaní systému, ktorý sú zadané ako use-case
- Sekvenčné diagramy (alebo diagramy spolupráce) v UML sa používajú na modelovanie interakcie medzi objektmi.

Structured analysis and design

- Štruktúrna analýza a dizajnové metódy zahŕňajú systémové modelovanie ako neoddeliteľnú súčasť metódy
- Metódy definujú množinu modelov, čo je proces pre odvodenie týchto modelov a pravidlá a pokyny, ktoré by mali platiť pre modely.
- CASE nástroje podporujú modelovanie ako súčasť štruktúrovanej metódy.

Method weaknesses (nedostatky metódy)

- nemodelujú non-funkčné požiadavky na systém
- Nemajú zvyčajne zahŕňať informácie o tom, či metóda je vhodná pre daný problém
- Môže produkovať príliš veľa dokumentácie
- Systémové modely sú niekedy príliš podrobne a zložité pre užívateľa na porozumenie

CASE workbenches (stoly???)

- Ucelený súbor nástrojov, ktorý je určený na podporu činnosti súvisiacich softvérových procesov, ako je analýza, návrh a testovanie. Môže byť začlenená do IDE ako Eclipse
- Podpora modelovania počas oboch požiadaviek inžinierstva a návrhu systému
- UML stoly poskytujú podporu pre všetky UML diagramy a obvykle nejaké automatické generovanie kódu. Java, C++, alebo C# môžu byť generované

Analysis workbench components (Komponenty analýzy pracovného stola)

- schéma editory
- Model analýzy a kontrolné nástroje
- Repository a súvisiace dopytovacie jazyky
- dátový slovník
- Správa o definícii a nástroje pre generovanie
- Nástroje na formovanie definícii
- Import/export prekladatelia
- Nástroje na generovanie kódu

kľúčové body

- Sémantické dátové modely môžu popísať logickú štruktúru údajov, ktoré sa importujú do alebo exportujú zo systémov
- Modely objektov popisujú logické systémové jednotky, ich členenie a agregácie
- Sekvenčné modely ukazujú, interakcie medzi aktérmi a systémovými objektmi, ktoré používajú
- Štruktúrované metódy poskytujú rámec pre rozvoj systémových modelov.

Architectural Design

Ciele: predstaviť architekt. návrh a diskutovať o jeho význame, vysvetliť rozhodnutia architekt. návrhu kt. museli byť urobené ,zaviesť 3 doplnkové architekt. štýly (organizácia, rozklad ,riadenie)

Softvérová architektúra: dizajn proces na identifikáciu podsystémov , ktoré tvoria systém a rámec podsystému kontroly a komunikácie je **architekt. dizajn.** Výsledkom tohto dizajn procesu je popis **softvérovej architektúry.**

Architektonické riešenie: skorá fáza návrhu systému, spojenie medzi špecifikáciou a dizajnom procesov, často sa vykonáva súbežne s inými špecifikovanými činnosťami, zahŕňa identifikáciu hlavných systémových komponentov a ich komunikácie

Výhody explicitnej architektúry: komunikácia so zúčastnenými stranami, systémová analýza, široká možnosť znovupoužitia

Architektúra a systémové charakteristiky: výkon, zabezpečenie, bezpečnosť, dostupnosť, udržovateľnosť

Konflikty architektúry: použitím large-grain(asi veľkozrnný) komponentov zvýšime výkon ale znížime udržiavateľnosť, redundantné dátá zlepšujú dostupnosť ale robia bezpečnosť zložitejšou

Systémové členenie: systém je rozdelený do interaktívnych podsystémov, architekt. dizajn - vyjadrený ako bloková schéma predstavujúca prehľad štruktúry systému, špecifické modely ukazujú ako podsystémy zdieľajú dátá

Box a line diagramy: veľmi abstraktné, užitočné pre komunikáciu a plánovanie projektu

Architektonické rozhodnutie o návrhu: architektonický dizajn je tvorivý proces, takže proces sa lísi v závislosti od typu vyvíjaného systému rad spoločných rozhodnutí pokrýva všetky konštrukčné procesy.

Znovupoužitie architektúry(reuse): Systémy v rámci rovnakej domény majú často podobné architektúry, ktoré odrážajú doménové koncepty, linky aplikácie produktov sú postavené okolo jadra architektúry s variantmi, ktoré spĺňajú konkrétnie požiadavky zákazníkov.

Architektonické štýly: architektonický model systému môže byť v súlade s všeobecným architektonickým modelom alebo štýlu, väčšina veľkých systémy sú heterogénne a nenasledujú jednu architektúru

Architektonické modely: **statický štrukturálny model**, ktorý ukazuje hlavné súčasti systému, **dynamický model**, ktorý zobrazuje procesnú štruktúru systému, **rozhranie model**, ktorý definuje pod-systém rozhrania, **vzťahy model**, ako je model dátového toku, ktorý ukazuje, pod-systém vzťahov, **distribučný model**, ktorý ukazuje, ako sú čiastkové systémy, rozmiestnené vo počítačoch.

Systém organizácie: odráža základnú stratégiu, ktorá sa používa pre štruktúru systému, tri organizačné štýly sú široko používané: **1.štýl zdieľaného uložiska dát** **2.štýl zdieľania služieb a serverov** **3.štýl zvrstvený alebo štýl abstraktných strojov**

Úložisko modelu: Podsystémy musia vymieňať dátu. To možno vykonať dvoma spôsobmi:

a) Zdieľané údaje sú držané v centrálnej databáze a môžu byť prístupné všetkým podsystémom

b) Každý podsystém udržiava svoje vlastné databázy a dát explicitne odovzdáva ostatným podsystémom.

-výhody: efektívny spôsob ako zdieľať veľké množstvo dát, zdieľaný model je vydávaný ako úložisko schémy

-nevýhody: dátová evolúcia je nákladná a zložitá, nevyhnutný kompromis medzi podsystémami, ťažko distribuovať efektívne

Client-Server Model: distribuovaný systém model, ktorý ukazuje, ako sú dátá a ich spracovanie distribuované v celej rade komponentov

-výhody: jednoduchá distribúcia dát, efektívnosť pri využívaní zariadení v sieti, jednoduchý upgrade serverov alebo pridanie nového

-nevýhody: redundantné riadenie v každom serveri, žiadny centrálny register mien a služieb, neefektívna výmena dát medzi podsystémami

Abstract Machine model: používa sa na modelovanie prepojenia jednotlivých systémov, organizuje systém do sady vrstiev, každá z nich poskytuje súbor služieb, podporuje inkrementálny vývoj podsystémov v rôznych vrstvách

KeyPoints

Softvérová architektúra je základný rámec pre štruktúrovanie systému. Architektonická rozhodnutia zahŕňajú rozhodnutia o aplikačnej architektúre, distribúcii a architektonické štýly, ktoré majú byť použité.

Rôzne architektonické modely, ako sú structural model, a control model and a decomposition model môžu byť vytvorené.

Systémové organizačné modely zahŕňajú úložisko modelov, klient server modely a abstraktné modely strojov.

Architectural Design 2

Modulárne rozdelenie štýlov- štýly rozdeľujúce podsystémy do modulov, nie je žiadny pevný rozdiel medzi organizáciou systému a modulárnym rozdelením.

Podsystémy a modely- podsystém= systém vo svojom vlastnom systéme, ktorého prevádzka je nezávislá na službách poskytovaných inými podsystémami, **modul**= je súčasťou systému, ktorý poskytuje služby iným komponentom, ale aby normálne neboli považované za samostatný systém.

Modulárne rozdelenie: štrukturálna úroveň, kde sú podsystémy rozdelené do modulov

Dva modulárne modely rozdelenia zahŕňajú: a) Objektový model, kde sa systém rozdelí do interakcie objektu b) Potrubie alebo model dátového toku, kde sa systém rozdelí do funkčných modelov, ktoré transformujú vstupy na výstupy.

Objektové modely: usporiadavajú systém do súboru voľne previazaných objektov s dobre definovanými rozhraniami

-výhody: objekty sú voľne viazané, čo umožňuje, že môžu byť upravené bez ovplyvnenia ďalších objektov, objekty môžu odrážať entity reálneho sveta,OO jazyky sú široko používané

Funkciovo orientované prúdové spracovanie(Function-oriented pipelining): Funkčné transformácie spracovávajú svoje vstupy za účelom produkcie výstupov, môže byť označené ako pipe a filter mode. Varianty tohto prístupu sú veľmi časté. Keď sú transformácie sekvenčné, to je sekvenčný model, ktorý sa vo veľkej miere používa v systémoch spracovania dát.

-výhody: intuitívna organizácia pre komunikáciu so zainteresovanými stranami, jednoduché pridanie nových transformácií, jednoduché realizovať buď ako súbežný alebo sekvenčný systém

Control style: týkajú sa toku riadenia medzi subsystémami. Odlišný od modelu systému rozdelenia. Centralizované riadenie+ Event-based riadenie.

Centralizované riadenie: Kontrolný podsystém preberá zodpovednosť za riadenie vykonávania ostatnými subsystémami. Call-return(zhora nadol) model + Manager model(súbežnosť systémov)

Udalosťami riadené systémy: Hnaný externé generovanými udalosťami, kt. načasovanie presahuje kontrolu podsystémov kt. riadia udalosti. Dve principal event-driven modely:

a) Broadcast models: Účinný pri integrácii subsystémov na rôznych počítačoch v sieti. Subsystémy zaregistrujte záujem o konkrétnu udalosť. Ak sa tieto príznaky vyskytnú, je kontrola prevedená na podsystému, ktorý dokáže spracovať udalosť.
b) Interrupt-driven models. Používa sa v real-time systémoch, kde rýchla odozva na udalosť je nevyhnutná. Sú známe typy prerušenia definované pre každý typ. Každý typ je spojený s lokáciou pamäte a prepnutia hardwaru spôsobí prevod do jeho jednotky.

Reference Architecture: Architektonické modely môžu byť špecifické pre určité domény aplikácie. 2 typy domain-specific model:

a) Generic models- bottom-up models

b) Reference models - top-down models.

Môžu byť použité ako základ pre zavedenie systému, alebo pre porovnanie rôznych systémov. Pôsobí ako štandard, podľa ktorej môžu byť hodnotené systémy.

Key Points

Modulárne rozdelenie Modelov zahŕňa object models a pipelining models.
Kontrolné modely zahŕňajú centralised control a event-driven models.
Referenčné architektúry môžu byť použité pre komunikáciu s konkrétnou doménou architektúry a posúdiť a porovnať architektonické návrhy.

Kapitola 7

Objektovo-orientovaný návrh

Objektovo-orientovaný vývoj- objektovo orientované analýzy(OOA), dizajn(OOD) a programovanie(OOP)sú príbuzné, ale odlišné.

-OOA sa zaoberá vývojom objektového modelu aplikačnej domény.

-OOD sa zaoberá vývojom objektovo-orientovaného systémového modelu na implementáciu požiadaviek.

-OOP sa zaoberá realizáciou OOD pomocou OO programovacieho jazyka, ako je Java, C ++ alebo C #.

Charakteristiky OOD- Objekty sú abstrakcie reálnych alebo systémových entít, ktoré spravujú samy seba, sú nezávislý, zapuzdrený stav a reprezentácia informácií. Funkčnosť systému je vyjadrená v súvislosti s objektovými službami. Zdieľané dátové oblasti sú eliminované. Objekty spolu komunikujú zasielaním správ. Objekty môžu byť distribuované a môžu byť realizované postupne alebo súbežne.

Objektovo-orientované modelovanie- Neoddeliteľnou súčasťou objektovo-orientovaného vývoja je vývoj UML modelov predstavujúcich nejaký systém. Poznáme:

-štrukturálne a behaviorálne UML modely (boli popísané v predchádzajúcich prednáškach o modelovaní)

-typy diagramov použité v tejto prednáške sú z skôr z UML 1 ako UML 2, ale rozdiely sú minimálne

Objektovo-orientovaný dizajnový proces- Štruktúrované dizajnové procesy zahŕňajú vývoj a niekoľko rôznych systémových modelov. Vyžadujú veľa úsilia pre vývoj a udržiavanie týchto modelov a, pre malé systémy, to nemusí byť cenovo efektívne. Avšak, pre veľké systémy vyvinuté rôznymi skupinami sú dizajnové modely základným komunikačným mechanizmom.

Procesné fázy- Hlavné aktivity v akomkoľvek OO dizajne zahŕňajú:

- Kontext: definícia kontextu a spôsoby používania systému
- Architektúra: Návrh architektúry systému;
- Objekty: Identifikácia hlavných systémových objektov
- Modely: Vývoj dizajnových modelov
- Rozhranie: Špecifikácia objektových rozhraní

Príklad-Opis systému počasia:

Sú to počasie mapujúce systémy nevyhnutné pre vytvorenie mapy počasia na základe údajov získaných zo vzdialených, nenavštevovaných meteorologických staníc a iných zdrojov dát, ako sú pozorovateľne počasia, balóny, satelia.

Meteorologické stanice odovzdávajú svoje údaje do oblastného počítača ako reakciu na žiadosť z tohto počítača. Oblastný počítačový systém overuje zhromaždené dáta a integruje ich s dátami z rôznych zdrojov. Integrované údaje sú archivované a, na základe údajov z archívu a digitalizovanej databáze máp, je vytváraný súbor lokálnych meteorologických máp. Mapy môžu byť vytlačené pre distribúciu na špeciálnej tlačiarne alebo môžu byť zobrazené v niekoľkých rôznych formátoch.

Postup:

1.) Context:

-vývoj porozumenia vzťahov medzi navrhovaným softvérom a jeho vonkajším prostredím

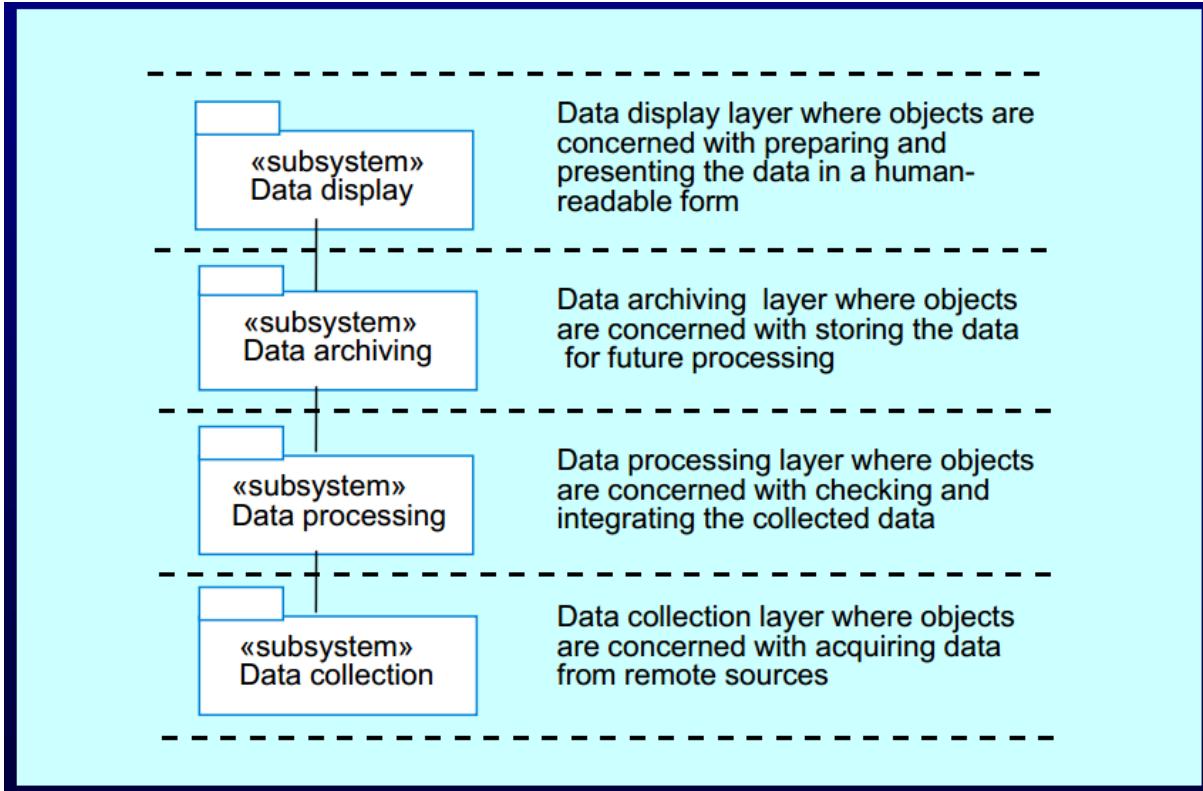
-systémový kontext- statický model, ktorý opisuje ďalšie systémy v prostredí.

Vhodné je použiť subsystémový model pre zobrazenie ďalších systémov.

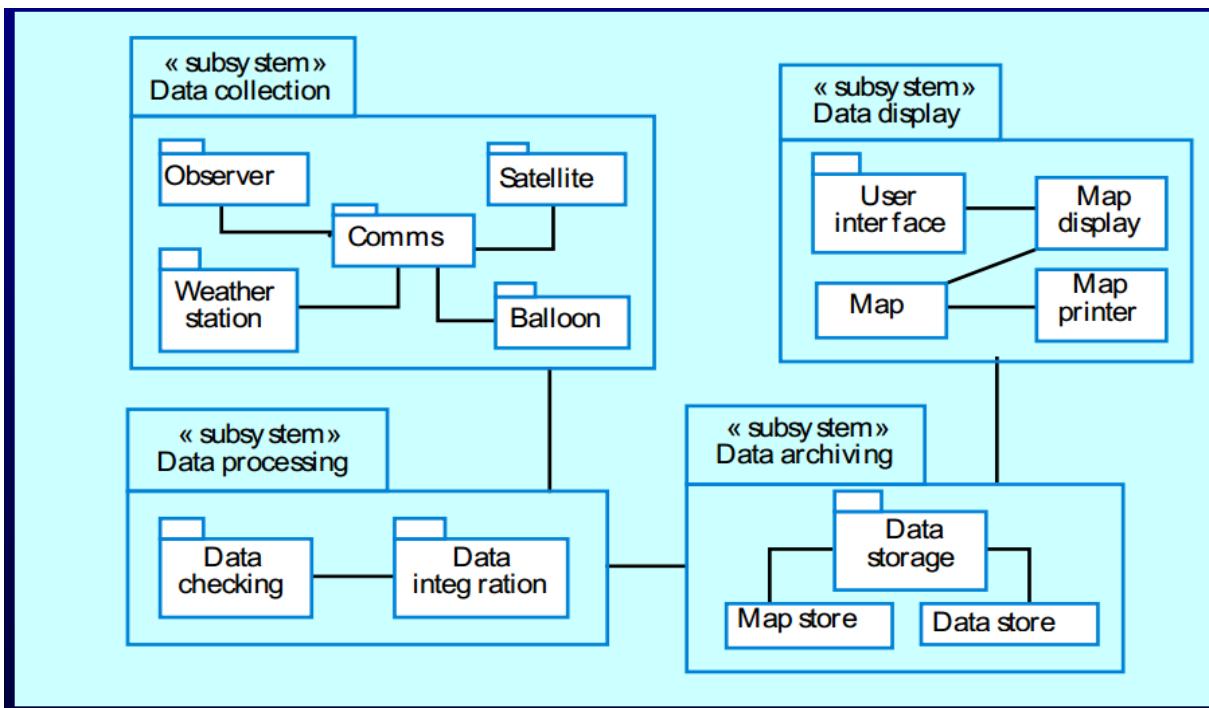
Nasledujúci snímka ukazuje systémov na celom Meteorologické stanice systému.

-model použitia systému- dynamický model, ktorý popisuje, ako systém interaguje s jeho prostredím. Pre zvýraznenie interakcií sú použité use-case-y.

2.) Navrstvená architektúra:

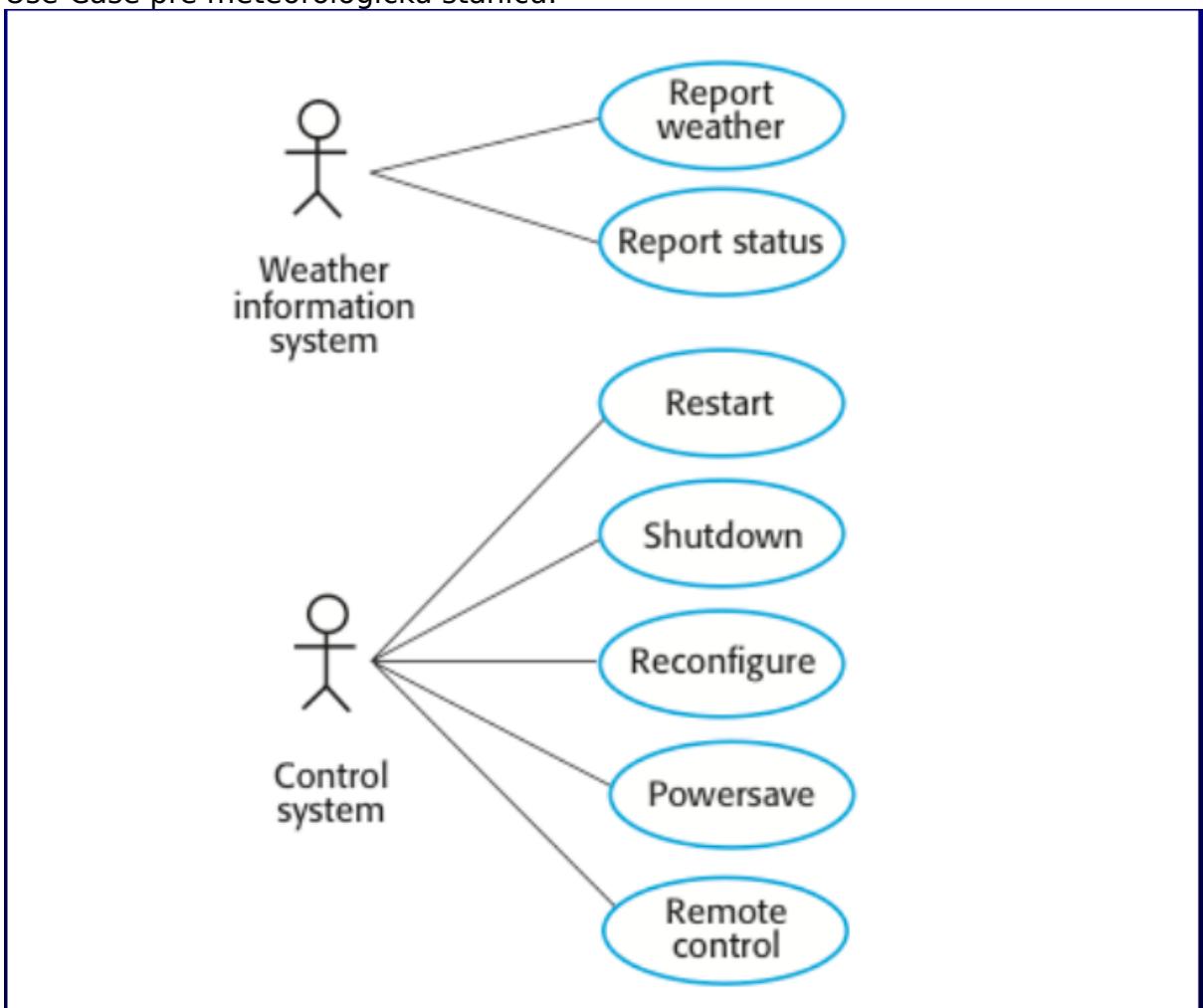


1. Data display- dáta, ktoré zobrazujú vrstvu, kde sú objekty zamerané na prípravu a prezentáciu dát do pre človeka zrozumiteľnej formy
2. Dat archiving- vrstva, ktorá uchováva dáta, kde sú objekty zamerané na uchovávanie dát pre budúce využitie
3. Data processing- vrstva, v ktorej sú objekty zamerané na kontrolu a integráciu nazhromaždených dát
4. Data collection-vrstva, ktorej objekty sú zamerané získavanie dát zo vzdialených zdrojov



Use-case modely- Use-case modely sú používané na reprezentáciu každej interakcie so systémom, systémové funkcie vystupujú ako elipsy a interagujúce entity(Actors) ako fugúrky.

Use-Case pre meteorologickú stanicu:



Popis:

System	Weather station
Use case	Report weather
Actors	Weather information system, Weather station
Description	The weather station sends a summary of the weather data that has been collected from the instruments in the collection period to the weather information system. The data sent are the maximum, minimum, and average ground and air temperatures; the maximum, minimum, and average air pressures; the maximum, minimum, and average wind speeds; the total rainfall; and the wind direction as sampled at five-minute intervals.
Stimulus	The weather information system establishes a satellite communication link with the weather station and requests transmission of the data.
Response	The summarized data is sent to the weather information system.
Comments	Weather stations are usually asked to report once per hour but this frequency may differ from one station to another and may be modified in the future.

Use-Case: správa o počasí

Actors: Meteorologický informačný systém, meteorologická stanica

Popis: M. stanica posiela údaje o počasí do m.i.s. Údaje zaznamenávajú maximum, minimum a priemer teploty na zemskom povrchu a vo vzduchu, tlaku vzduchu, rýchlosťi vetra, celkovú hodnotu zrážok a smer vetra v 5-minútových intervaloch.

Stimul: M.i.s. je s m.s. spojený satelitnou komunikačnou linkou a pomocou nej posiela žiadosť o prenos dát.

Odpoveď: M.s. posiela vyhodnotené dátá m.i.s.

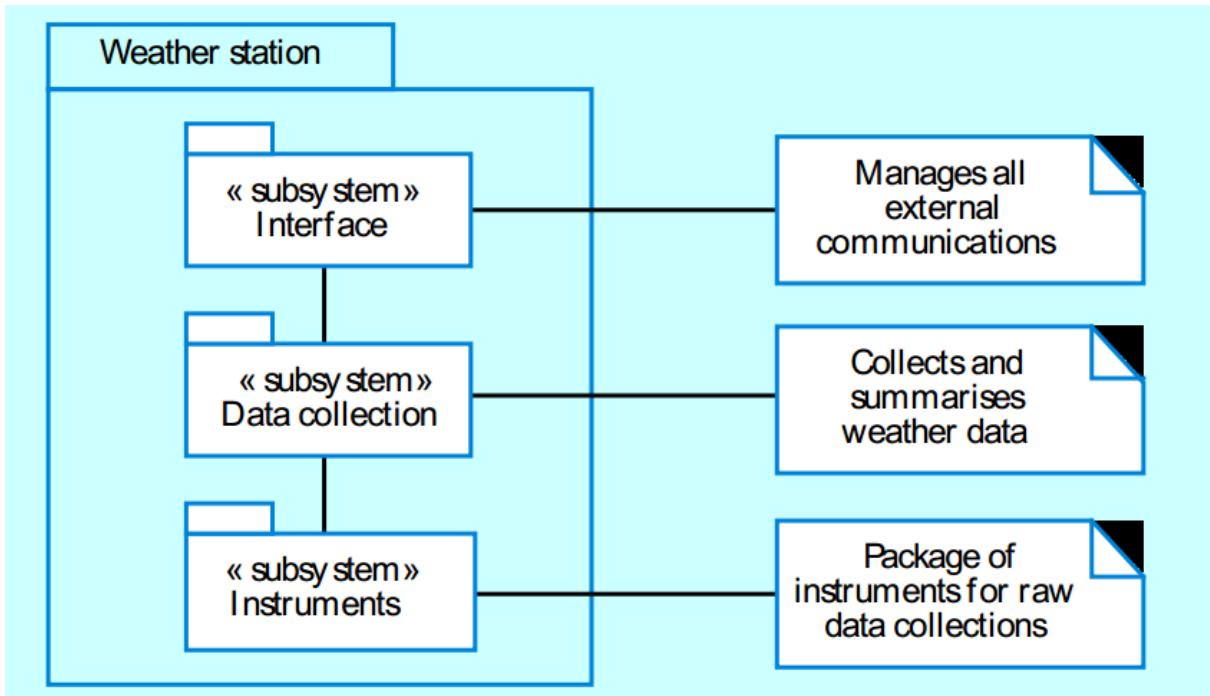
Komentáre: M.s. väčšinou podávajú správu raz za hodinu, ale táto frekvencia sa môže lísiť od stanice k stanici a môže byť neskôr zmenená.

Architektúra- Akonáhle sú interakcie medzi systémom a jeho prostredím pochopené, môžete tieto informácie použiť pre návrh architektúry systému.

Vrstvené architektúry, ako je popísané v kapitole 11 sú vhodné pre meteorologické stanice, lebo:

- majú vrstvu Interface pre riadenie komunikácie;
- majú vrstvu Data-Collection pre správu nástrojov;
- majú vrstvú Instruments pre zber dát.

Normálne by nemalo byť v architektonickom modeli viac než 7 entít.



Objekty- Identifikácia objektov (alebo tried objektov) je najťažšia časť OOD. Neexistuje žiadny čarovný recept pre identifikáciu objektu. Systémový dizajnéri sa opierajú o zručnosti, skúsenosti a znalosti. Identifikácia objektov je iteratívny proces. Je nepravdepodobné, aby sa to začiatočník naučil hned' na prvýkrát.

Prístupy k identifikácii- Použite gramatický prístup založený na prirodzenom jazykovom opise systému (používa sa v Hoodovej OOD metóde). Identifikáciu založte na hmatateľných objektoch v aplikačnej doméne. Použite behaviorálny prístup a identifikujte objekty na základe toho, čoho sa zúčastňujú a ako sa správajú. Použite analýzu založenú na situácií- objekty, atribúty a metódy v identifikujte v každej situácii.

Opis meteorologickej stanice- balík softvérom riadených nástrojov, ktoré zhromažďujú dátá, niektoré dátá spracovávajú a posielajú ich na ďalšie spracovanie. Medzi nástroje patria vzdušné a pozemné teplomery, anemometer, veterník, barometer a zrážkomer. Údaje sú zhromažďované periodicky. Ak je vydaný povel k prenosu informácií o počasí, meteorologická stanica spracuje a zosumarizuje zhromaždené údaje, ktoré sú následne po obdržaní žiadosti prenesené do mapovacieho počítača.

Triedy objektov stanice:

- Pozemné teplomery, Anemometer, Barometer- "Hardware" objekty súvisiace s nástrojmi v systéme.
- Meteorologicke stanice- Základné rozhranie meteorologickej stanice s jej Prostredím, odráža interakcie identifikované v use-case modely.
- Údaje o počasí- Zapuzdržujú súhrnné údaje z prístrojov.

Kľúčové pojmy:

- OO vývoj zahŕňa prijatie OO prístupu vo všetkých fázach od špecifikácie až po programovanie
- OO návrh zahŕňa návrhnutie systému pomocou objektov ako základnú abstrakciu a reprezentuje systém ako spojený súbor modelov v UML
- konštrukcia OO proces zahŕňa niekolko stupňov: kontext, architektúra a objekty

Kapitola 7

Objektovo-orientovaný dizajn2

Ďalšie objekty a objektové vylepšenia:

- Použite znalosti o doméne na identifikáciu viac objektov a operácií.
- Meteorologické stanice by mali mať jedinečný identifikačný kód
- Meteorologické stanice sú situované vzdialene, takže chyby prístrojov musia byť hlásené automaticky. Preto sú potrebné atribúty a operácie pre samokontrolu.
- Aktívne alebo pasívne objekty - v tomto prípade, objekty sú pasívne a zhromažďujú údaje radšej na žiadosť než samostatne. To zvyšuje flexibilitu na úkor doby spracovania regulátora.

Triedy objektov:

WeatherStation	WeatherData
identifier	
reportWeather ()	airTemperatures
reportStatus ()	groundTemperatures
powerSave (instruments)	windSpeeds
remoteControl (commands)	windDirections
reconfigure (commands)	pressures
restart (instruments)	rainfall
shutdown (instruments)	
	collect ()
	summarize ()

Ground thermometer	Anemometer	Barometer
gt_Ident	an_Ident	bar_Ident
temperature	windSpeed	pressure
get ()	windDirection	height
test ()	get ()	get ()
	test ()	test ()

Dizajnové modely:

Dizajnové modely zobrazujú objekty, ich triedy a vzťahy medzi týmito entitami.
-Statické modely popisujú statickú štruktúru systému, pokiaľ ide o triedy objektov a vzťahy.

-Dynamické modely popisujú dynamické interakcie medzi objektmi.

Príklady dizajnových modelov:

Sub-systémové modely, ktoré zobrazujú logické zoskupenie objektov do súdržných subsystémov.

-Sekvenčné modely - zobrazujú sled objektov a interakcie medzi nimi

-Stavové modely - zobrazujú, ako jednotlivé objekty zmenia svoj stav pri reakcii na udalosť

-Ostatné modely sú use-case modely, agregačné modely, generalizácie atď.

Subsystémové modely:

-zobrazujú, ako je návrh rozdelený do logicky súvisiacich skupín objektov.

- V UML sú tieto objekty zobrazené ako balíky- zapuzdrené konštrukcie
- na označenie balíkov ako subsystémov sa používa bežná UML notácia.

Rozklad subsystémov (A):

Subsystémy rozhraní:

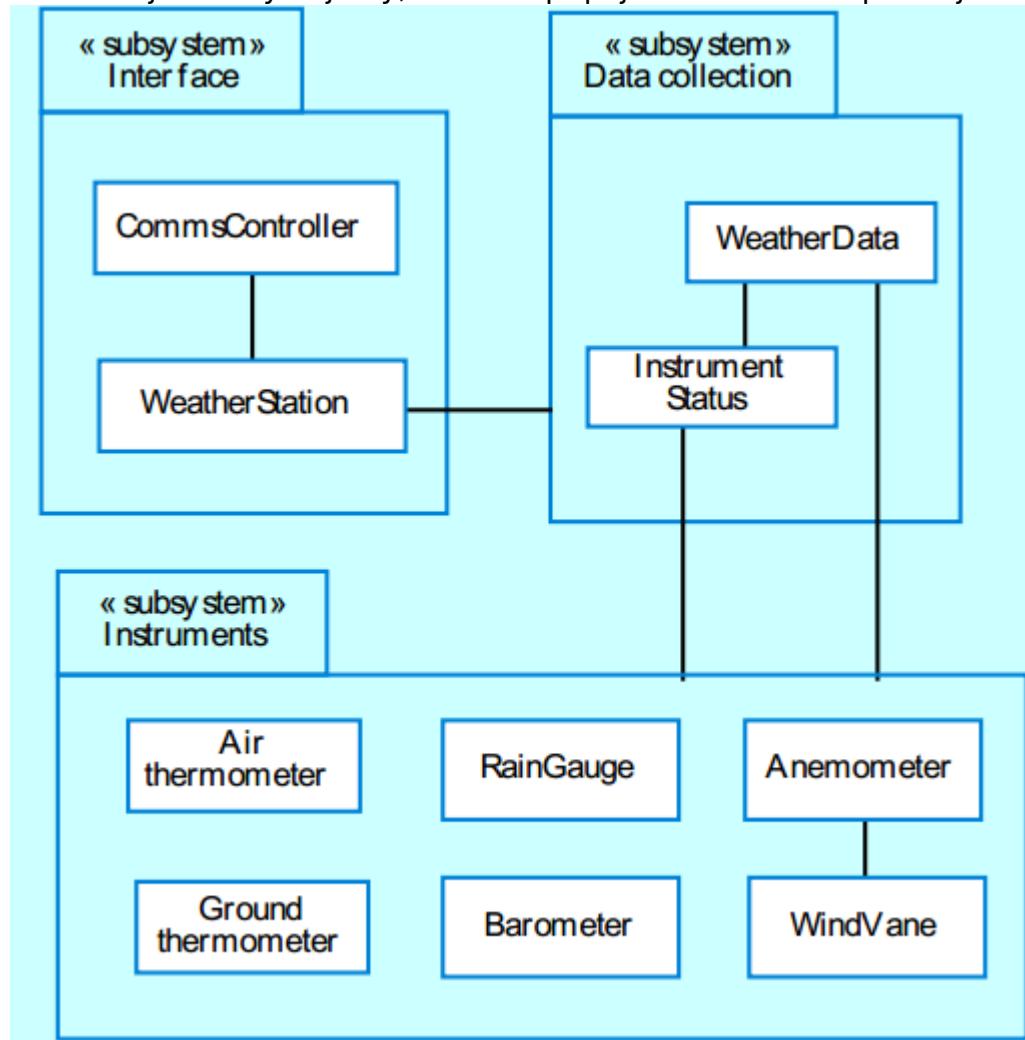
- Obsahujú objekty v systéme, ktoré sa zaobrajú prepojením meteorologickej stanice s externými systémami
- Môže obsahovať iné objekty od tu zobrazených, napr. užívateľské rozhranie pre testovanie.

Subsystémy dátového zberu:

- Obsahujú objekty, ktoré implementujú stratégie pre zber dát
- Sú zámerne oddelené od aktuálnych kolekcií dát kvôli umožneniu zmien týchto stratégíí

Subsystémy nástrojov:

- Obsahujú všetky objekty, ktoré sú pripojené k hardvéru prístrojov

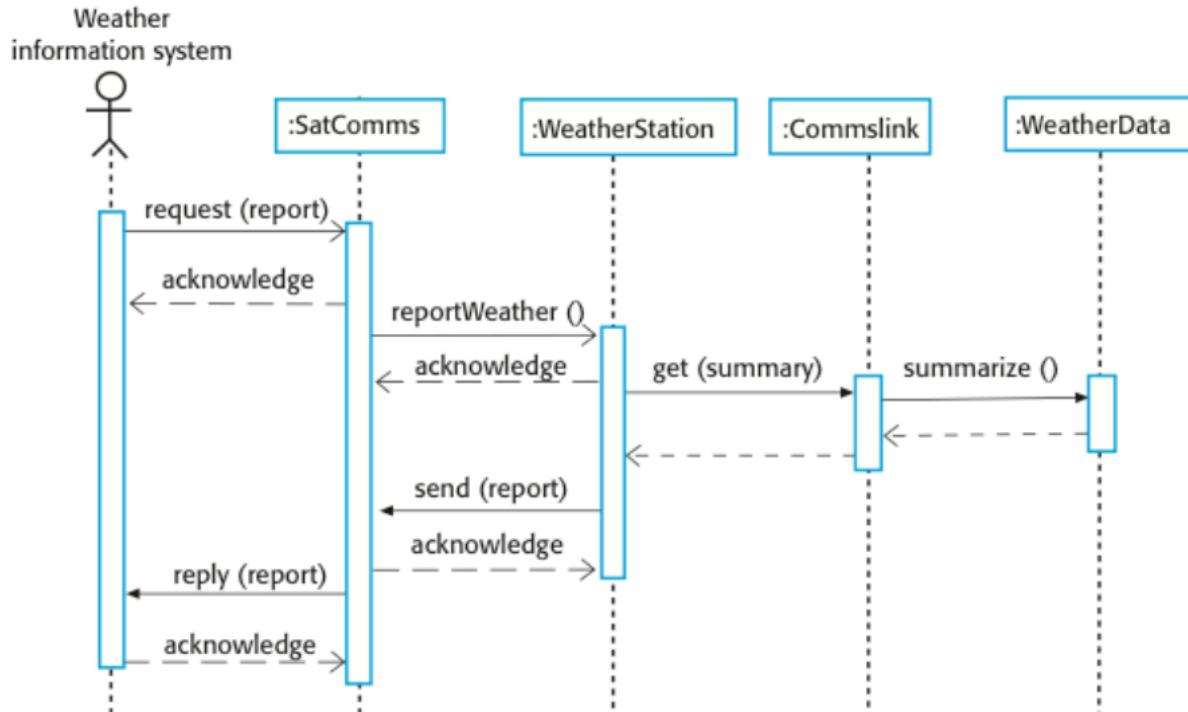


Sekvenčné modely:

Sekvenčné modely ukazujú sled interakcií medzi objektami

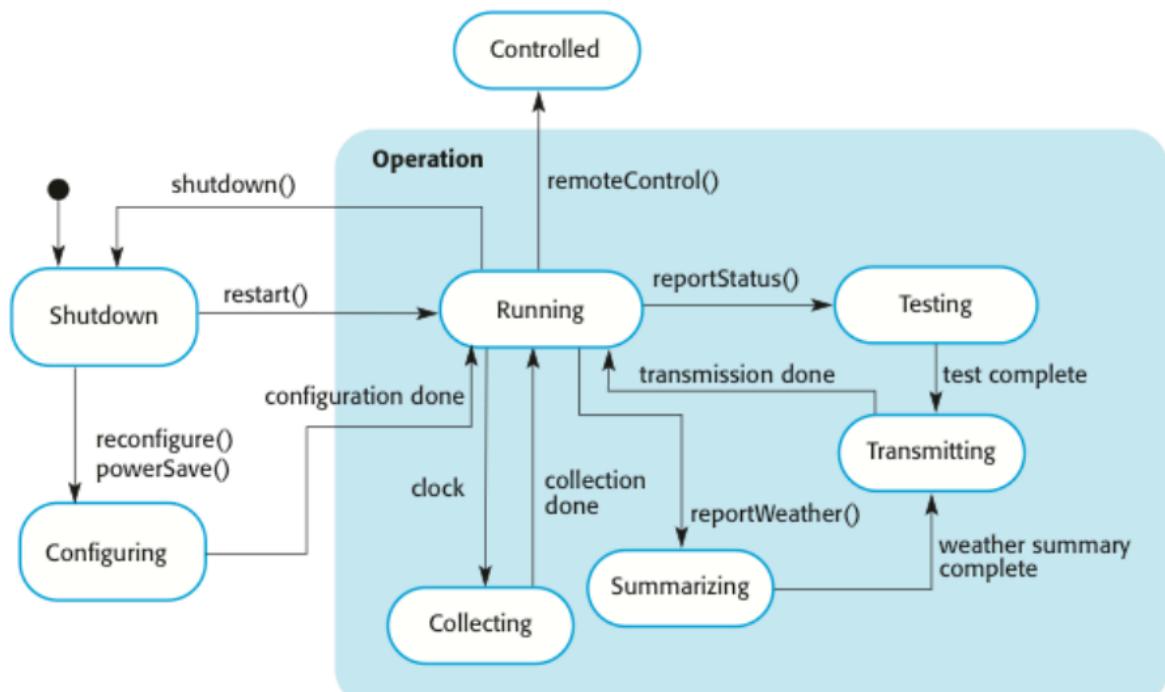
- Objekty sú usporiadané horizontálne , zhora
- Čas je reprezentovaný vertikálne, takže modely treba čítať zhora nadol
- Interakcie sú zastúpené šípkami, rôzne typy šípok predstavujú rôzne typy interakcií
- tenký obdĺžnik v objektovej life-line predstavuje čas, kedy je objekt aktuálne interagujúcim objektom v systéme

Sled zberu dát:



Stavový graf:

- Zobrazuje, ako objekty reagujú na rôzne prevádzkové požiadavky a na prechody medzi stavmi vyvolanými týmito žiadosťami
- Stavy sú reprezentované ako zaoblené obdĺžníky
- prechody medzi stavmi sú vyznačené šípkami medzi obdĺžnikmi



Stavy systému meteorologickej stanice:

- Ak stav objektu je „vypnutý“ potom reaguje na reconfigure () hlásenie
- If reportWeather (), ide systém do Wummarisingstate
- Zberný stav nastane, keď je prijatý hodinový signál

Rozhrania:

- Objektové rozhranie musí byť určené tak, aby objekty a ďalšie komponenty mohli byť navrhnuté paralelne
- Projektanti by sa mali vyhnúť navrhovaniu implementácie rozhraní a ukrývať ich v objektoch
- Objekty môžu mať niekoľko rozhraní (???which are viewpoints on the methods provided???)
- UML používa pre špecifikáciu rozhraní diagramy tried, načo sa dá použiť ale aj Java

```
interface WeatherStation {  
  
    public void WeatherStation () ;  
  
    public void startup () ;  
    public void startup (Instrument i) ;  
  
    public void shutdown () ;  
    public void shutdown (Instrument i) ;  
  
    public void reportWeather ( ) ;  
  
    public void test () ;  
    public void test ( Instrument i ) ;  
  
    public void calibrate ( Instrument i ) ;  
  
    public int getID () ;  
  
} //WeatherStation
```

Vývoj návrhu:

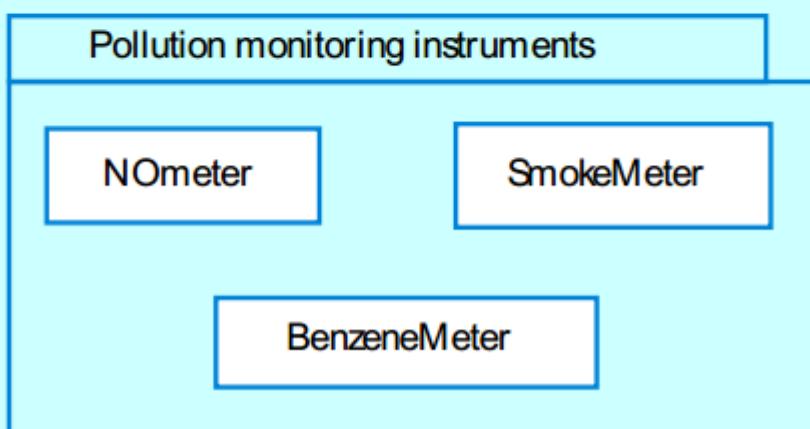
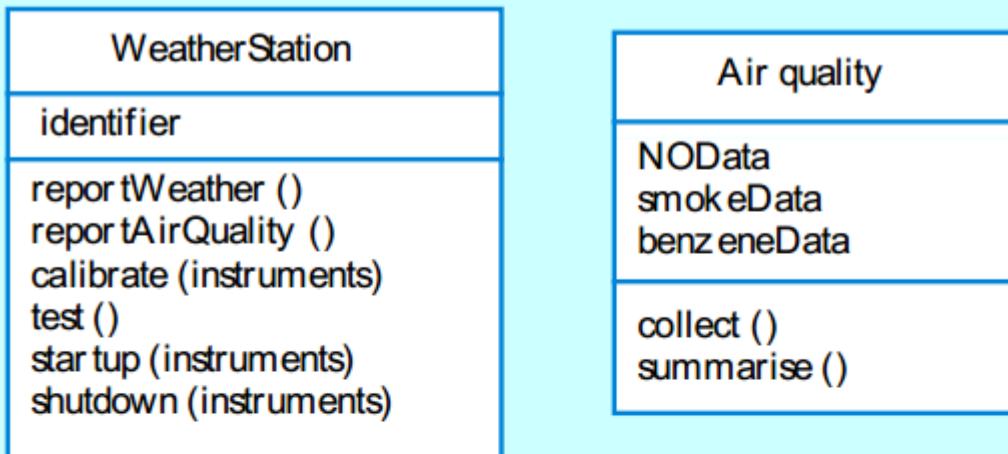
- Skrytie informácií vo vnútri objektov znamená to, že zmeny vykonalé na objekte neovplyvnia nepredvídateľným spôsobom ostatné objekty.
- Predpokladajme, že zariadenie pre monitoring znečistenia ovzdušia sú pridávané do meteorologických staníc. Zo vzoriek vzduchu a vypočítavajú objem rôznych znečistujúcich látok v ovzduší.

-Tieto hodnoty sú prenášané s údajmi o počasí

Zmeny, ktoré to vyžaduje:

- Pridať triedu objektov s názvom Air quality ako časť WeatherStation.
- Pridať operáciu reportAirQuality do WeatherStation. Upraviť ovládací softvér, aby zbierał údaje o znečistení
- Pridať objekty predstavujúce nástroje pre monitoring znečistenia ovzdušia.

Monitoring znečistenia:



Klúčové pojmy:

Pri objektovo orientovanom navrhovanacom procese môžu byť vyrobené rôzne modely. Patria sem statický a dynamický systémový model.

-objektové rozhrania by mali byť presne definované, použitím napr. programovacieho jazyka Java

-Objektovo orientovaný návrh potenciálne zjednodušuje evolúciu systému.

Rapid software development (prednáška 9-1)

Ciele

- Ak chcete vysvetliť, ako iteratívny, inkrementálny vývojový proces vedie k rýchlejsiemu dodaniu viac užitočného softvéru
- Na diskutovanie podstaty agilných vývojových metód
- Pre vysvetlenie role prototypovania v softvérovom procese
- Vysvetliť princípy a praktiky extrémneho programovania

Rýchly vývoj software (Rapid software development)

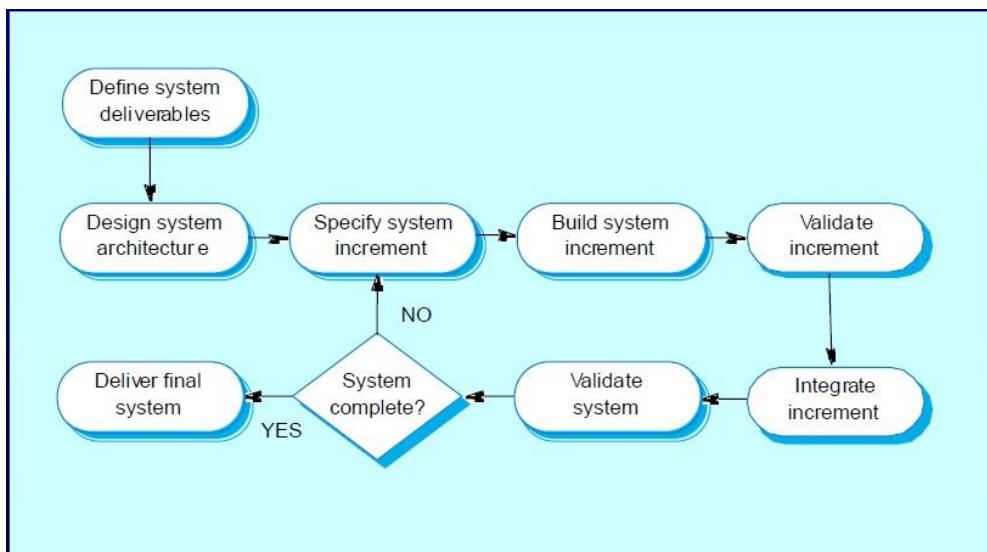
- Vzhľadom k rýchlo sa meniacemu podnikateľskému prostrediu, podniky musia reagovať na nové príležitosti a súťaže.
- To si vyžaduje, aby softvér a rýchly vývoj a dodania nie je často najdôležitejšou požiadavkou pre softvérové systémy.
- Podniky môžu byť ochotné akceptovať nižšiu kvalitu softvéru ak je to možná rýchla dodávka základných funkcií.

Požiadavky

- Vzhľadom k meniacim sa prostrediam, je často nemožné, aby sa dospelo na stabilnú, konzistentnú sadu systémových požiadaviek.
- Preto je model vodopádového vývoja nepraktický a prístup k rozvoju založenom na iteratívnej špecifikácii a dodávky je jediný spôsob, ako dodávať softvér rýchlo.

Charakteristika RAD procesov

- Procesy špecifikácie, návrhu a implementácie sú súbežné. Neexistuje žiadna podrobná špecifikácia a projektová dokumentácia je minimálna.
- Systém je využívaný v sérii krokov. Koncoví užívatelia zhodnotia každý prírastok a podávajú návrhy neskorších stupňov.
- Užívateľské rozhranie systému je zvyčajne využívané pomocou interaktívneho vývojového systému



Výhody kumulatívneho vývoja

- **Urýchlenie zákazníckych služieb.** Každý prírastok prináša najvyššiu prioritu funkčnosti k zákazníkovi.
- **Užívateľ je zapojený do systému.** Používatelia musia byť zapojení do vývoja, čo znamená, že je viac pravdepodobné, že sa splnenia ich požiadavky a užívatelia sú viac spojení so systémom.

Problémy s kumulatívne vývoj

- **problémy riadenia** • Pokrok môže byť ľažké posúdiť a problémy, ľažko nájsť, pretože neexistuje žiadna dokumentácia na preukázanie, čo sa stalo.
- **zmluvné problémy** • Normálna zmluva môže obsahovať špecifikáciu, bez špecifikácie, rôzne formy zmluvy musia byť použité.
- **problémy validácie** • Bez špecifikácie, na čo je systém testovaný?
- **problémy údržby** • Neustále zmeny majú tendenciu k poškodeniu softvérovú štruktúru, robiť to drahšie, meniť a vyvíjať na splnenie nových požiadaviek.

agilné metódy

- Nespokojnosť nákladov, ktoré sú v návrhových metódach viedie k vytvoreniu agilných metód. Tieto metódy:
 - Zamerajte sa na kód, skôr než navrhovanie;
 - Sú založené na iteratívnom prístupe k vývoju softvéru;
 - Sú určené na rýchlu dodávku pracovných programov a vyvíjať ich tak rýchlo, aby splňali meniace sa požiadavky.
- Agilné metódy sú pravdepodobne najvhodnejšie pre malé/stredné obchodné systémy alebo počítačové produkty

Rýchly vývoj aplikácií

- Agilné metódy získali veľkú pozornosť, ale mnoho rokov boli používané aj iné prístupy k rýchlemu vývoju aplikácií.
- Tie sú určené na rozvoj dátovo náročných podnikových aplikácií a spoľahnúť sa na programovanie a prezentovanie informácií z databázy.

RAD nástroje prostredia

- Databázový programovací jazyk
- rozhranie generátor
- Odkazy na kancelárské aplikácie
- report generátory

generácie rozhrania

- Mnohé aplikácie sú založené na zložitých formánoch a rozvoj týchto foriem ručne je časovo náročná činnosť.
- RAD prostredie zahŕňajú podporu pre generovanie obrazovky, vrátane:
 - Interaktívne definície formulára pomocou drag and drop techniky;
 - Forma prepojenia, kde je uvedené poradie formulárov, ktoré sa majú predložiť;
 - Overenie formulára, kde je definovaný povolený rozsah v poliach formulára.

vizuálne programovanie

- Skriptovacie jazyky, ako je Visual Basic podporujú vizuálne programovanie, kde je prototyp vyvinutý na vytvorenie používateľského rozhrania zo štandardných položiek a združenie komponentov s týmito položkami
- Veľká knižnica komponentov existuje podporujúca tento typ rozvoja
- Tie môžu byť prispôsobené tak, aby vyhovovali špecifickým požiadavkám danej aplikácie

Problémy s vizuálnym vývojom

- Ľažké koordinovať rozvoj tímovej práce.
- Žiadne explicitná architektúra systému.
- Komplexné závislosti medzi časťami programu môžu spôsobiť problémy udržiavateľnosti.

COTS reuse

- Efektívny prístup k rýchlemu vývoju znamená nakonfigurovať a prepojiť existujúce časti mimo regálových systémov (???:D)
- Napríklad, systém pre správu požiadaviek by mohol byť vytvorený pomocou:
 - Databázy pre ukladanie požiadaviek;
 - Textový procesor pre zachytenie požiadaviek a správy vo formáte;
 - Tabuľka pre správu sledovateľnosti;

Software prototyping

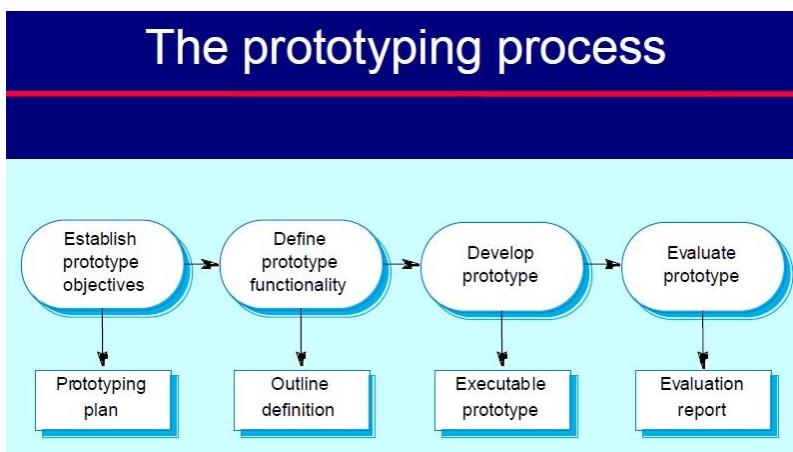
- Prototyp je pôvodná verzia systému používaného na preukázanie pojmu a vyskúšanie možnosti dizajnu.
- Prototyp môže byť použitý v:
 - Požiadavky inžinierstva proces (requirements engineering process), ktorý pomôže s požiadavkami elicítáciu a overovanie;
 - V návrhových procesoch, aby preskúmala možnosti a vytvoriť dizajn používateľského rozhrania;
 - V procese testovania spustiť back-to-back testov.

Konfliktné ciele

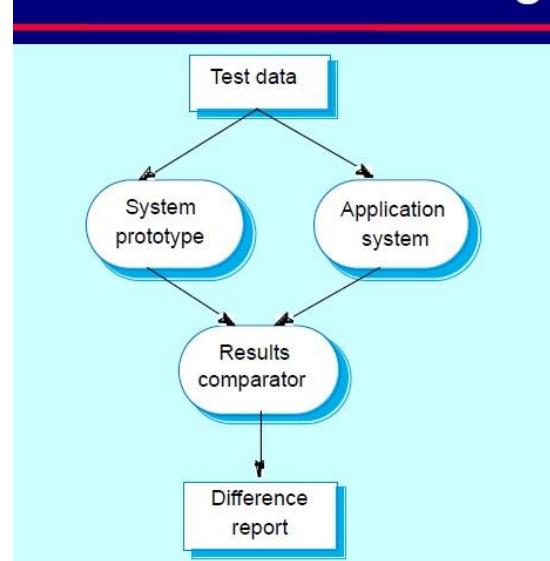
- Cieľom incremental development je dodať funkčný systém koncovým užívateľom. Vývoj začína s požiadavkami, ktoré sú najlepšie pochopené.
- Cieľom throw-away prototyping je overenie požiadaviek na systém. Proces prototyping začína s požiadavkami, ktoré sú zle pochopené.

Výhody prototypovania

- Vylepšená použiteľnosť systému.
- Bližší zápas na skutočné potreby užívateľov.
- Lepšia kvalita prevedenia.
- Lepšia udržiavateľnosť.
- Znižené rozvojové úsilie.



Back to back testing



Jednorazové (throw-away) prototypy

- Prototypy sa musia zlikvidovať po rozvoji, pretože nie sú dobrým základom pre produkčný systém:
 - Môže byť nemožné naladiť systému a splniť non-funkčné požiadavky;
 - Prototypy sú zvyčajne dokumentované;
 - Štruktúra prototypu je zvyčajne znižená prostredníctvom rýchlych zmien;
 - Prototyp pravdepodobne nebude spĺňať bežné štandardy organizačnej kvality.

kľúčové body

- Iteratívny prístup k vývoju softvéru vedie k rýchlejšiemu dodaniu softvéru.
- Agilné metódy sú iteračné metódy rozvoja, ktoré majú za cieľ zníženie nákladov na rozvoj rézii, a tak vytvárať softvér rýchlejšie.
- Rýchle prostredie pre vývoj aplikácií sú databázové programovacie jazyky, nástroje generácie forma (form generation tools) a odkazy na kancelárske aplikácie.
- Prototyp throw-away sa používa k objavovaniu požiadaviek a dizajnové možnosti.

Extreme Programming

agilné metódy

- Nespokojnosť nákladov, ktoré sú v návrhových metódach viedie k vytvoreniu agilných metód. Tieto metódy:
 - Zamerajte sa na kód, skôr než navrhovanie;
 - Sú založené na iteratívnom prístupe k vývoju softvéru;
 - Sú určené na rýchlu dodávku pracovných programov a vyvíjať ich tak rýchlo, aby spĺňali meniace sa požiadavky.
- Agilné metódy sú pravdepodobne najvhodnejšie pre malé/stredné obchodné systémy alebo počítačové produkty

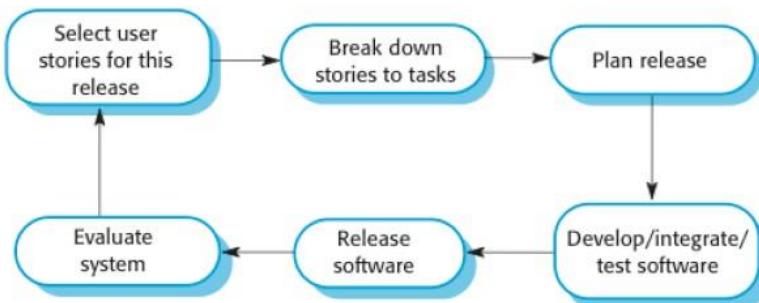
Problémy agilných metód

- Môže byť ťažké udržať záujem zákazníkov, ktorí sa podieľajú na procese.
- Členovia tímu môžu byť nevhodní pre intenzívne zapojenie, ktoré charakterizuje agilné metódy.
- Uprednostnenie zmeny môže byť ťažké, ak existuje viac zúčastnených strán.
- Zachovanie jednoduchosti vyžaduje prácu navyše.
- Zmluvy - môže byť problém ako s inými prístupmi k iteratívному vývoju.

Extrémne programovanie

- Snáď najznámejšia a najpoužívanejšia agilná metóda.
- Extrémne programovanie (XP) má "extrémny" prístup pre iteratívny vývoj.
 - Nové verzie môžu byť postavené niekoľkokrát za deň;
 - Prírastky sú dodávané zákazníkom každé 2 týždne;
 - Všetky skúšky musia byť spustené pre každú verziu a build je prijatá iba vtedy, pokiaľ testy prebehnú úspešne.

The XP release cycle



Extreme programming practices 1 a 2 – ceknut v prednaske kto by chcel

XP a agilné princípy

- Inkrementálny vývoj je podporovaný prostredníctvom malých, častých verzií systému.
- Zapojenie zákazníkov znamená angažmán na plný úväzok s tímom.

- Ľudí nemožno spracovať pomocou páŕ programov - kolektívne vlastníctvo a proces, ktorý sa vyhýba dlhým pracovným hodinám.
- Zmena podporovaná prostredníctvom pravidelných verzií systému.
- Zachovanie jednoduchosti neustálym refactoring kódu.

zapojenie zákazníka

- Zapojenie zákazníka je kľúčovou súčasťou XP, kde zákazník je súčasťou vývojového tímu.
- Úloha zákazníka je:
 - Ak chcete pomôcť rozvíjať príbehy, ktoré definujú požiadavky
 - Ak chcete pomôcť priority funkcie, ktoré majú byť realizované v každej verzii
 - Ak chcete pomôcť rozvíjať akceptačné testy, ktoré posudzujú, či systém splňa jeho požiadavky.

požiadavky scenára

- V XP sú užívateľské požiadavky vyjadrené ako scenáre alebo užívateľské príbehy.
- Tie sú napísané na kartách a vývojový tím to používa na plnenie úloh. Tieto úlohy sú základom plánu a odhadu nákladov.
- Zákazník si vyberá príbehy pre zaradenie do budúceho vydania na základe svojich priorit a odhadov plánu.

Story card for document downloading – prednaska

XP a zmena

- Konvenčná múdrost v softvérovom inžinierstve je navrhnuť zmeny. Stojí za to stráviť čas a úsilie na predvídanie zmien, pretože to znižuje náklady neskôr v životnom cykle.
- XP však tvrdí, že to nie je užitočné lebo zmeny nemožno spoľahlivo predpokladať.
- Skôr sa navrhuje neustále zlepšovanie kódu (refactoring), aby sa menilo ľahšie, keď má byť zmena vykonaná.

Refactoring

- Refactoring je proces zlepšovania kódu, kde je kód reorganizovaný a prepísaný, aby bol efektívnejší, zrozumiteľnejší, atď
- Refactoring je nutný, pretože časté zmeny znamenajú, že kód je vyvinutý postupne, a preto má tendenciu byť chaotickým.
- Refactoring by nemalo meniť funkčnosť systému.
- Automatizované testovanie zjednodušuje refactoring, lebo môžete zistiť, či zmenený kód stále beží, ak testy úspešne.

Testovanie v systéme Windows XP

- Test-first vývoj.
- Inkrementálny vývoj testu od scenárov.
- Zapojenie užívateľov v oblasti vývoja a validácie testu.
- Automatizované testovacie sa používa na spustenie všetkých čiastkových skúšok zakaždým, keď je nová verzia postavená.

Task cards for document downloading – prednaška

Test case description – prednaška

Test-first vývoj

- Písanie testov pred kódom upresňuje požiadavky, ktoré majú byť realizované.

- Testy sú písané ako programy skôr ako údaje tak, aby mohli byť vykonané automaticky. Test zahŕňa kontrolu, že kód bol správne vykonaný.
- Všetky predchádzajúce a nové testy sú automaticky spustené, keď je pridaná nová funkcia. Preto sa skontrolujte, či do novej funkcie nebola zavedená chyba.

Pár programovanie (Pair programming)

- V XP programátori pracujú vo dvojiciach, sedia spolu na vývoji kódu.
- To pomáha rozvíjať spoločné vlastníctvo kódu a šíriť vedomosti naprieč tímom.
- Slúži ako proces neformálne preskúmania, že na každý riadok kódu sa pozrela viac ako 1 osoba.
- Povzbudzuje refactoring lebo celý tím môže ľažiť z toho.
- Merania ukazujú, že produktivita vývoja s párom programovania je podobná ako u dvoch ľudí, ktorí pracujú nezávisle na sebe.

Problémy s XP

- zapojenie zákazníka
 - To je snáď najťažší problém. Môže byť ľažké alebo nemožné nájsť zákazníka, ktorý môže zastupovať všetky zúčastnené strany, a ktorý môžu byť priatý mimo svoju bežnú prácu, aby sa stal súčasťou tímu XP. Pri generických produktoch nie je žiadny "zákazník"- marketingový tím nemusí byť typom skutočného zákazníka

Problémy s XP

- architektonické riešenie
 - Inkrementálny štýl vývoja môže znamenať, že nevhodné architektonické rozhodnutia sú v ranej fáze procesu.
 - Problémy to nemusí byť jasné, keď boli vykonané mnohé funkcie a refactoring architektúra je veľmi drahá.
- skúšobné uspokojenie
 - tím verí, že preto, že má veľa testov, systém je správne testovaný.
 - Vzhľadom k automatizovanému testovaciemu prístupu, je tu tendencia k vývoju testov, ktoré možno ľahko automatizovať, ako testy, ktoré sú "dobré" testy.

kľúčové body

- Extrémne programovanie zahŕňa praktiky, ako napríklad systematické testovanie, neustále zlepšovanie a zapojenie zákazníkov.
- Zákazník sa podieľa na rozvoji požiadavky, ktoré sú vyjadrené ako jednoduché scenáre.
- Prístupom k testovaniu v XP je najmä pevnosť, kde sú spustiteľné testy vyvinuté pred napísaným kódom.
- Kľúčové problémy s XP sú problémy na získanie reprezentatívnych zákazníkov a problémy architektonického návrhu.

Testing A

Proces testovania

Testovanie Komponentov

- Testovanie individuálnych časti programu
- Testovanie má väčšinou na starosti tvorca komponentu (výnimkou sú niekedy kritické systémy)
- Testy sú vytvorené na základe tvorcových skúseností

Systémové testovanie

- Testovanie skupín komponentov spojených do systém alebo subsystém
- Zodpovednosť nezávislého testovacieho tímu
- Testy sú založené na systémových špecifikáciách

Testovanie komponentov	-----→	Systémové testovanie
Tvorca software		Nezávislý testovací tím

Chybové testovanie

- Ciel chybového testovania je zistiť chyby v programe
- Úspešný chybový test je test ktorý spôsobí neobvykle správanie programu
- Test ukáže prítomnosť nie absenciu chyby (defektu)

Ciele testovacieho procesu

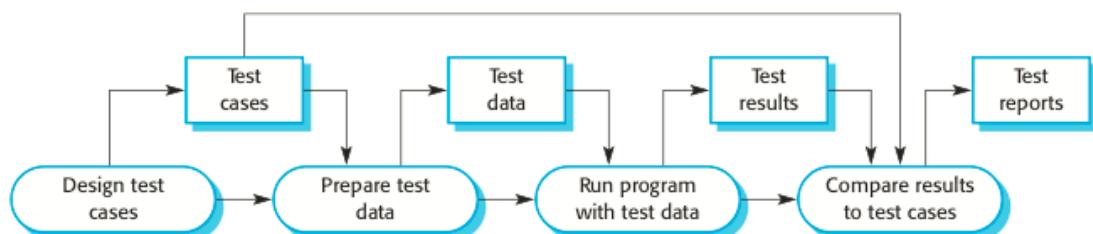
Validačné testovanie

- Demonstrovať tvorcovi a zákazníkovi, že softvér splňuje požiadavky
- Úspešný test, že systém pracuje ako má.

Chybové testovanie

- Zistenie chýb a kazov v softvéri, kde správanie programu nie je správne alebo nie je v súlade so špecifikáciou.
- Úspešný test je test ktorý donúti systém pracovať nesprávne a tým pádom odhalí chybu v systéme.

Proces testovania softvéru



Pravidlá testovania

- Iba dôkladné testovanie môže dokázať, že je program bez chýb. Ale dôkladné testovanie nie je možné.
- Pravidlá testovania definujú prístup používaný na výber systémových testov :
 - Všetky funkcie dostupné z menu by mali byť otestované
 - Kombinácie funkcií dostupných z rovnakého menu by mali byť otestované
 - Všade kde je potrebný vstup od užívateľa, musia byť otestované všetky funkcie so správnym aj nesprávnym vstupom.

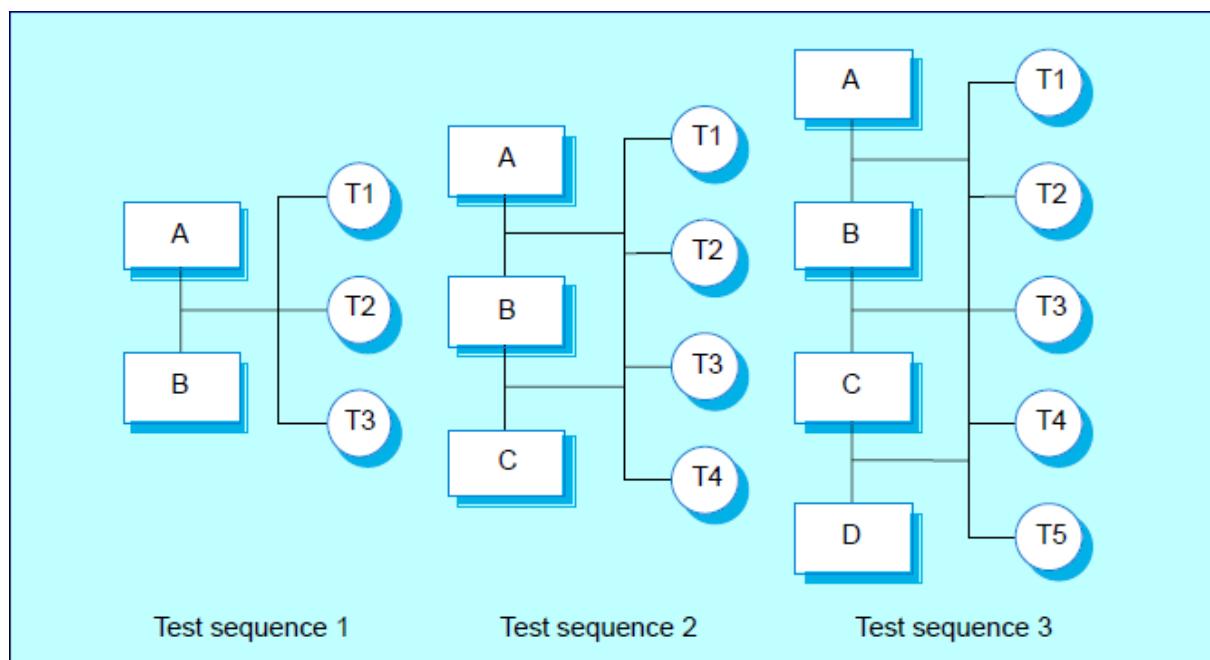
Systémové testovanie

- Zahŕňa zlučovanie komponentov na vytvorenie systému alebo subsystému.
- Môže zahŕňať testing an increment ktoré budú doručene zákazníkovi.
- Dve fázy :
 - Integračné testovanie – testovací tím má prístup k zdrojovému kódu systému.
Systém je testovaný počas zlučovania komponentov.
 - Release testing(uvolňovacie testovanie) – testovací tím testuje kompletný systém ktorý bude doručený ako čierna skrinka(black-box).

Integračné testovanie

- Zahŕňa vytváranie systému z jeho komponentov a testovanie týchto komponentov aby sa zistil problém ktorý môže vzniknúť z interakcie komponentov.
- Top – down integration
 - Vytvorí sa kostra systému a zaplní sa komponentami.
- Bottom – up integration
 - Integraruje infraštruktúru komponentov potom pridáva funkcionálne komponenty
- Na zjednodušenie lokalizácie chýb, systém by mal byť inkrementálne integrovaný.

Incremental integration testing



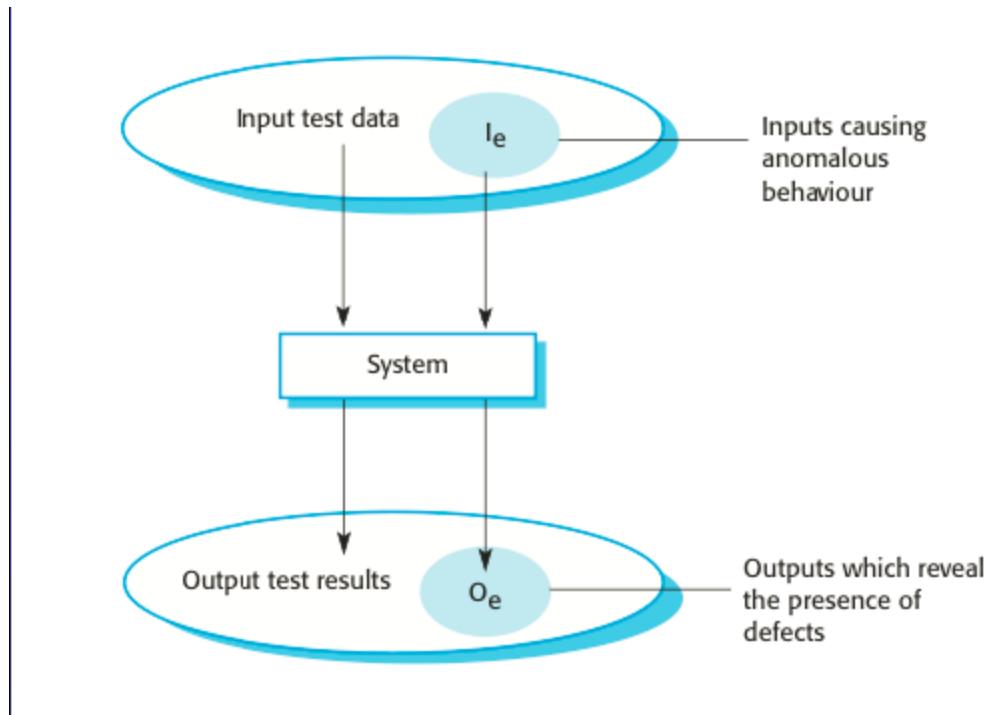
Testovacie prístupy

- Architektonická validácia
 - Top-down integration testing je lepšie v nachádzaní chýb v systémovej architektúre.
- Ukážka systému
 - Top-down integration testing dovoľuje limitovanú ukážku v počiatocných fázach vývoja.
- Implementácia testu
 - Často jednoduchšie s bottom-up integračným testom.
- Pozorovanie testu
 - Problém s oboma prístupmi. Je potrebný extra kód na pozorovanie testu.

Release testing(uvolňovacie testovanie)

- Proces testovania release systému, ktorý bude dodaný zákazníkovi.
- Primárny cieľ je zvýšiť istotu dodávateľa, že systém dodržuje požiadavky.
- Release testing je zvyčajne black-box alebo funkčné testovanie
 - Založené iba na systémových špecifikáciách
 - Testeri nevedia ako je systém implementovaný.

Black-box testing



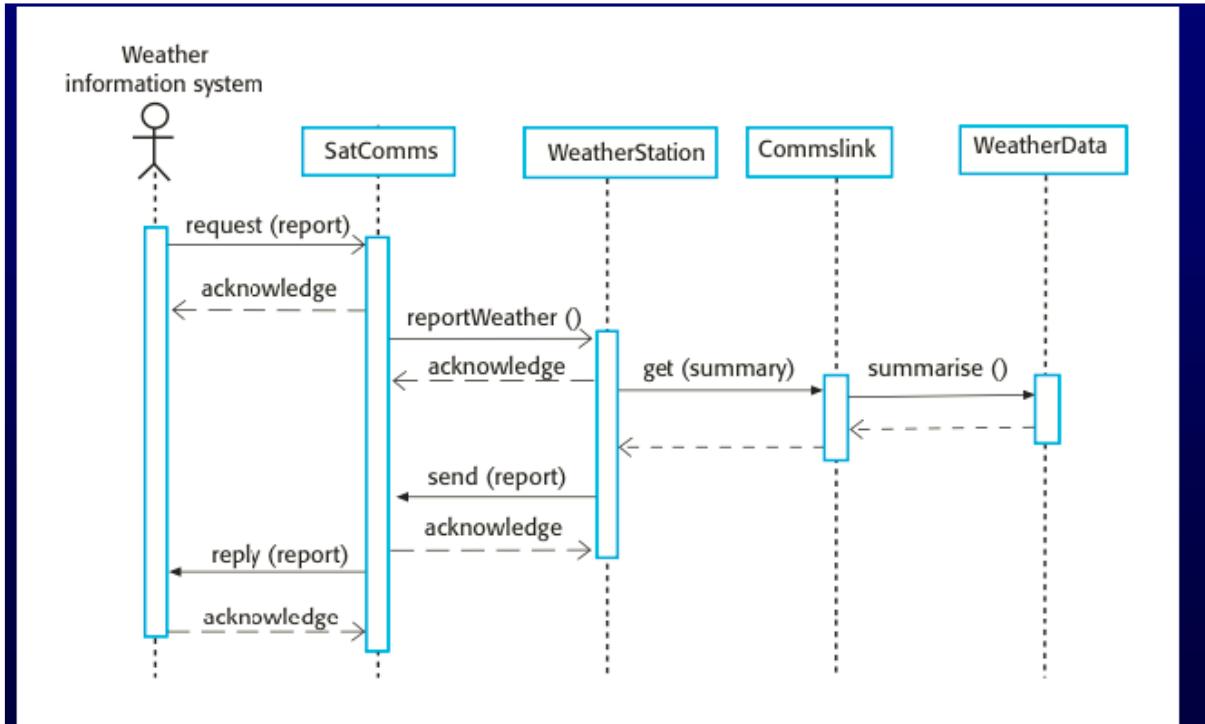
Smernice testovania

- Testovacie smernice sú pomôcky pre testovací tím, aby vybrali test, ktorý ukáže kazy systému
 - Vybrať vstupy ktoré nútia systém vygenerovať všetky chybové hlášky
 - Vytvoriť vstup ktorý spôsobí pretečenie buffera
 - Opakovať rovnaký vstup alebo vstup postupnosť niekoľko krát
 - Donútiť program aby vypísal nesprávne výstupy
 - Donútiť program aby výsledok výpočtu bol príliš krátky alebo príliš dlhý

Use cases

- Use cases môžu byť základ na vytváranie testov systému. Pomáhajú identifikovať operácie, ktoré majú byť testované a pomáhajú navrhovať potrebné testovacie cases.
- Zo sekvenčného diagramu caseu môžu byť identifikované vstupy a výstupy potrebné pre test.

Príklad sekvenčného diagramu pre dáky systém pre zbieranie dát zrážok



Testovanie výkonu

- Časť release testovania môže zahŕňať testovanie nevyhnutných vlastností systému ako výkon a spoľahlivosť.
- Test výkonu zväčša zahŕňa plánovanie sérií testov kde záťaž je stále zvyšovaná až kým je výkon systému neakceptovateľný.

Stress testing(Námahový test?)

- Preskúšava systém za jeho maximálneho dizajnovaného zaťaženia. Namáhanie systému často spôsobuje že vady výjdu na povrch.
- Namáhanie systému testuje správanie zlyhania...
Systém by nemal padnúť katastroficky. Stress(námahové) testovanie kontroluje neakceptovateľnú stratu dát alebo služieb.
- Námahové testovanie je hlavne dôležité pre distribúciu systémov, ktoré ukážu vážnu degradáciu pri preťaženej sieti.

Key points

- Testing can show the presence of faults in a system; it cannot prove there are no remaining faults.
- Component developers are responsible for component testing; system testing is the responsibility of a separate team.
- Integration testing is testing increments of the system; release testing involves testing a system to be released to a customer.
- Use experience and guidelines to design test cases in defect testing.

Testovanie softvéru

Testovanie komponentov

- Testovanie komponentov alebo jednotiek je proces testovania jednotlivých komponentov izolovane.
- Ide o proces testovania chýb.
- Komponentom môže byť:
 - Jednotlivé funkcie alebo metódy v rámci objektu;
 - Triedy objektu s niekoľkými atribútmi a metódami;
 - Zložené komponenty s definovanými rozhraniami používanými na prístup k ich funkcionalite.

Testovanie triedy objektu

- Kompletný test pokrycia triedy zahŕňa:
 - Testovanie všetkých operácií spojených s objektom;
 - Nastavovanie a vyskúšanie všetkých atribútov objektu;
 - Testovanie objektu vo všetkých jeho možných stavoch.
- Dedičnosť robí návrh triedy objektu zložitejším, pretože informácie nie sú lokalizované.

Rozhranie objektu meteostanice

WeatherStation	
identifier	
reportWeather ()	
reportStatus ()	
powerSave (instruments)	
remoteControl (commands)	
reconfigure (commands)	
restart (instruments)	
shutdown (instruments)	

Testovanie meteostanice

- Potrebujeme definovať testovacie prípady pre reportWeather, calibrate, test, startup and shutdown.
- Použitím stavového modelu identifikujeme sekvencie prechodov stavov na testovanie a sekvencie udalostí, ktoré spôsobujú tieto prechody.

- Napríklad:
 - Waiting -> Calibrating -> Testing -> Transmitting -> Waiting

Testovanie rozhraní

- Cieľom je nájsť chyby spôsobené chybami v rozhraniach alebo nesprávnymi predpokladmi o rozhraniach.
- Zvlášť dôležité pre objektovo orientovaný vývoj, keďže objekty sú definované ich rozhraniami.

Chyby rozhraní

- Nesprávne použitie rozhrania
 - Volajúci komponent volá ďalší komponent a spraví chybu v použití svojho komponentu (napr. parametre v nesprávnom poradí).
- Nerozumenie rozhraniu
 - Volajúci komponent vloží predpoklady o chovaní volaného komponentu, ktoré sú nesprávne.
- Chyby časovania
 - Volaný a volajúci komponent pracujú rôzne rýchlo a pristupované je k starým informáciám.

Pokyny pre testovanie rozhrania

- Navrhнемe testy tak, aby parametre volanej procedúry boli na extrémnych koncoch ich rozsahov.
- Vždy testujeme pointre parametrov z NULL pointrami.
- Navrhнемe testy, ktoré spôsobia pád komponentu.
- Použijeme stress testing v systémoch predávania správ.
- V systémoch so zdieľanou pamäťou, obmeňujeme poradie, v ktorom sú komponenty aktivované.

Návrh testovacích prípadov.

- Zahŕňa návrh prípadov testu (vstupy a výstupy) používané na testovanie systému.
- Cieľom návrhu testovacích prípadov je vytvoriť množinu testov, ktorá je efektívna v hodnotení a hľadaní chýb.
- Prístupy k návrhu:
 - testovanie založené na požiadavkách
 - testovanie častí
 - štruktúrované testovanie

Testovanie založené na požiadavkách

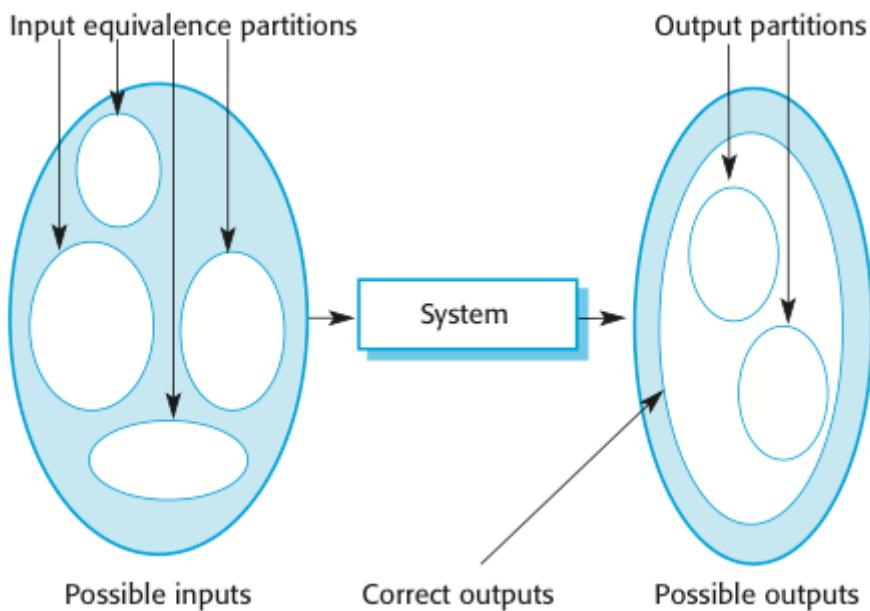
- Je technika, kde si zvážite každú požiadavku a odvodíte množinu testov pre túto požiadavku.
- Všeobecný princíp požiadavkového inžinierstva je, že požiadavky by mali byť testovateľné.

Testovanie časti

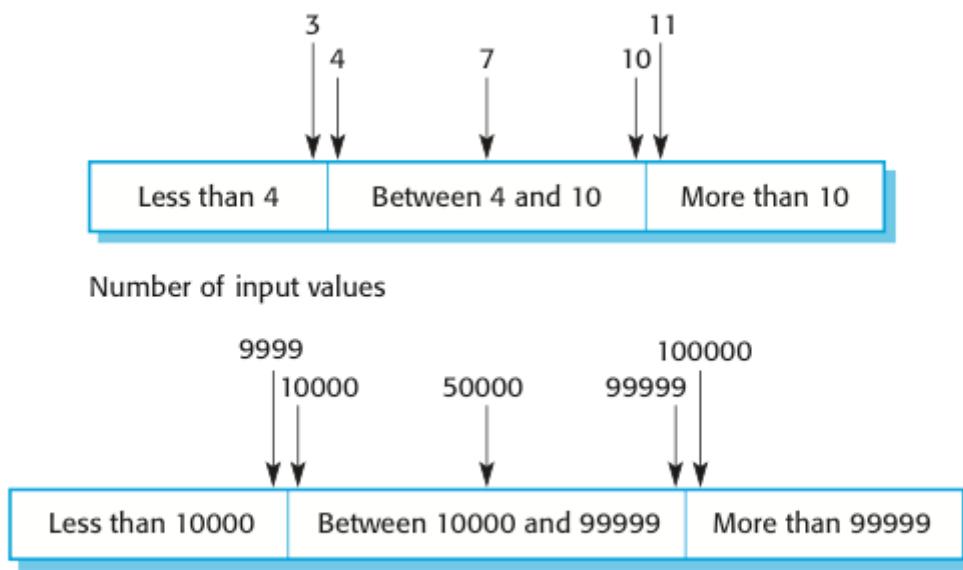
- Vstupné dáta a výstupné výsledky často spadajú do rôznych tried, kde všetky členy triedy sú príbuzné.

- Každá z týchto tried je oddiel ekvivalencie alebo domény, kde sa program chová rovnocenným spôsobom pre každého člena triedy.
- Testovacie prípady by mali byť vybrané z každého oddielu

Equivalence partitioning



Equivalence partitions



Špecifikácia vyhl'adávania

procedure Search (Key : ELEM ; T: SEQ of ELEM;
 Found : in out BOOLEAN; L: in out ELEM_INDEX) ;

Pre-condition

- the sequence has at least one element
- T'FIRST <= T'LAST

Post-condition

- the element is found and is referenced by L
(Found and T (L) = Key)

or

- the element is not in the array
(not Found and
not (exists i, T'FIRST >= i <= T'LAST, T (i) = Key))

Vyhľadávanie – oddiely vstupov

- Vstupy, ktoré spĺňajú predpoklady.
- Vstupy, ktoré nespĺňajú predpoklady.
- Vstupy, kde kľúčový prvok je členom poľa.
- Vstupy, kde kľúčový prvok nie je členom poľa.

Pokyny pre testovanie (sekvencie)

- Testujeme softvér so sekvenciami ktoré majú iba jednu hodnotu.
- Používame sekvencie rôznych veľkostí v rôznych testoch.
- Odvodť testy tak že prvé, stredné a posledné prvky sekvencie sú sprístupnené.
- Testujeme sekvenciami nulových dĺžok.

Štrukturálne testovanie

- Niekoľko nazývané white-box testovanie.
- Odvodenie testovacích prípadov v závislosti na štruktúre programu. Znalosť programu slúži na identifikáciu ďalších testovacích prípadov.
- Cieľom je vykonávať všetky príkazy programu (nie všetky kombinácie ciest).

Binárne vyhl'adávanie - equiv. partitions

- Predpoklady splnené, kľúčový prvok v poli.
- Predpoklady splnené, kľúčový prvok nie je v poli.
- Predpoklady nesplnené, kľúčový prvok v poli.
- Predpoklady nesplnené, kľúčový prvok nie je v poli.
- Vstupné pole má jednu hodnotu.
- Vstupné pole má párný počet hodnôt.
- Vstupné pole má nepárný počet hodnôt.

Testovanie cesty (Path testing)

- Cieľom testovania cesty je zabezpečiť, aby súbor testovacích prípadov je taký, že každá cesta cez program sa uskutoční aspoň raz.

- Východiskovým bodom pre testovanie cesty je graf prietoku programom, ktorý zobrazuje uzly reprezentujúce rozhodnutia a oblúky, ktoré predstavujú tok riadenia programu.
- Príkazy s podmienkami, sú teda uzly v grafe prietoku.

Nezávislé cesty

- 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 14
- 1, 2, 3, 4, 5, 14
- 1, 2, 3, 4, 5, 6, 7, 11, 12, 5, ...
- 1, 2, 3, 4, 6, 7, 2, 11, 13, 5, ...
- Testovacie prípady by mala byť stanovené tak, že všetky z týchto ciest sú vykonané
- Dynamický analyzátor programu môže byť použitý na overenie či boli vykonané všetky cesty.

Klúčové body

- Testovanie rozhraní je navrhnuté tak, aby sa dali objaviť chyby v rozhraniach kompozitných komponentov.
- Equivalence partitioning je spôsob zisťovania testovacích prípadov – všetky prípady v oddiele by sa mali správať rovnako.
- Štrukturálna analýza sa spolieha na analyzovanie programu a odvodenia testov z analýz.
- Automatizácia testu redukuje ceny testov podporov testovacieho procesu s radom softvérových nástrojov.

Pozn. som vyniechával také kokotiny v modrom, lebo to boli kokotiny :D si to pozrite v prednaske

10-1 Znovupoužite softvéru

Opäťovné použitie softvéru (system reuse)

- Vo väčšine technických odborov sú systémy navrhnuté tak, že sa skladajú z už existujúcich zložiek, ktoré boli použité v iných systémoch.
- Softvérové inžinierstvo bolo viac zamerané na originálny(pôvodný) vývoj, ale teraz sa skôr zameriava na dosiahnutie lepšieho, rýchlejšieho softvéru s nižšími nákladmi
- Preto musíme prijať návrhový proces, ktorý je založený na opäťovnom použití softvéru.

Opakované použitie na báze softvérového inžinierstva

- Aplikačný systém znovupoužitia
 - Celá časť systémovej aplikácie môže byť znova použitá buď začlenením bezo zmeny do iných systémov (COTS reuse), alebo vytvorením (vyvinutím) aplikačnej rodiny
- Opakované použitie súčasti(komponentov)
 - Súčasti(komponenty) aplikácie z čiastkových systémov môžu byť znova použité na jednotlivé objekty
- Opakované použitie predmetov a funkcií
 - Softvérové komponenty, ktoré implementujú jeden dobre definovaný objekt alebo funkciu môžu byť znova použité

Výhody znovupoužitia

- zvýšená spoľahlivosť - systém, ktorý už bol v praxi použitý a otestovaný je spoľahlivejší ako nový systém
- znížené riziká - Ak existuje softvér, tak je menšia neistota v nákladoch na opäťovné použitie tohto softvéru, ako v nákladoch na jeho nový vývoj
- efektívne využitie špecialistov - Namiesto toho, aby títo odborníci robili rovnakú prácu na rôznych projektoch, môžu títo špecialisti vyvinúť opakovane použiteľný softvér
- dodržiavanie štandardov - štandardy môžu byť implementované ako súbor štandardných znovupoužiteľných komponentov
- zrýchlený vývoj – Uvedenie systému na trh čo najskôr je často dôležitejšie ako celkové náklady na vývoj. Znovupoužitie softvéru môže urýchliť tvorbu systému, pretože aj vývoj a aj čas overenia by mal byť menší (zredukovaný)

Nevýhody znovupoužitia

- zvýšená cena údržby - ak nie je zdrojový kód znovupoužitého softvéru dostupný, cena údržby môže byť zvýšená , pretože opäťovne použité prvky systému môžu byť stále v rozpore so systémovými zmenami.
- nedostatočná podpora nástrojov - nástroje CASE nemusia podporovať vývoj znovupoužitia softvéru
- syndróm to-sme-my-nevymysleli - Niektorí softvéroví inžinieri niekedy radšej znova napíšu niektoré komponenty (časti), pretože sa domnievajú, že ich môžu zlepšiť na opakovane použiteľné komponenty. To súvisí čiastočne aj s dôverou a čiastočne aj s tým, že písanie originálneho softvéru je považovaná za náročnejšie ako opäťovné použitie softvéru od niekoho iného
- vytvorenie a údržba knižnice komponentov – na naplnenie a opakované používanie knižnice komponentov a zaistenia softvéru sú vysoké náklady. Naše súčasné techniky pre klasifikáciu, katalogizáciu a vyhľadávanie softvérových komponentov sú nezrelé (nie úplne vyvinuté).
- hľadanie, porozumenie a adaptácia komponentov - Softvérové komponenty musia byť objavené v knižnici, pochopené a niekedy aj prispôsobené pre prácu v novom prostredí. Inžinieri musia plne dôverovať, že daný komponent bude pracovať v novom prostredí

Prístupy k znovupoužitiu

- návrhové vzory - ukazujú abstraktné a konkrétné objekty a ich interakciu
- component-based vývoj - systém sa vyvíja integrovaním komponentov ktoré sú v súlade s normami
- aplikačný framework - kolekcia abstraktných a konkrétnych tried, ktoré môžu byť adaptované a rozšírené na vytváranie aplikačných systémov
- Legacy system wrapping(zabalenie dedičného systému(?)) - Staršie systémy, ktoré môžu byť "zabalené" tým, že sa definuje sada rozhraní a poskytuje sa prístup k týmto starším systémom prostredníctvom týchto rozhraní.
- systémy zamerané na služby - vývoj prepojením zdieľaných služieb, ktoré môžu byť externe dodávané
- Application product lines (aplikačná skupina výrobkov?) - Typ aplikácie je zovšeobecnený okolo spoločnej architektúry tak, aby mohol byť prispôsobený rôznym spôsobom pre rôznych zákazníkov.
- COTS integrácia - systém sa vyvíja integrovaním už existujúcich aplikačných systémov
- konfigurovatelné vertikálne aplikácie (?) - generický systém navrhnutý tak, aby sa dal nakonfigurovať na splnenie konkrétnych požiadaviek zákazníka
- programové knižnice - knižnice tried a funkcií ktoré sú bežne používané sú k dispozícii pre opakované použitie
- programové generátory - vkladajú znalosti jednotlivých typov aplikácií a dokážu generovať systémy alebo ich fragmenty v danej doméne
- hlădiskovo orientovaný vývoj softvéru - zdieľané komponenty sú "tkané" do aplikácie na rôznych miestach, počas komplikácie programu

Faktory plánovania znovupoužitia

- vývojový rozvrh pre softvér
- predpokladaná životnosť softvéru
- pozadie, zručnosti a skúsenosti tímu
- kritickosť soft.systému a jeho nefunkcionálne požiadavky
- aplikačné domény
- výkonná platforma softvéru

Koncept znovupoužitia

- Ked' znovupoužívame program alebo jeho konštrukčné prvky, musíme nasledovať návrh vykonaný pôvodným vývojárom daného komponentu (časti)
- To môže obmedziť príležitosti na opäťovné použitie.
- Avšak, viac abstraktnou formou opäťovného použitia je koncept znovupoužitia, kde je konkrétny postup popísaný nezávislým spôsobom implementácie
- Dva hlavné prístupy k opakovanému použitiu koncepcie sú:
 - **Návrhový vzor (design pattern)**
 - **Generatívne programovanie**

Návrhový vzor (design pattern)

- je to znovupoužitie abstraktných vedomostí o probléme a jeho riešení
- vzor by mal byť dostatočne abstraktný aby mohol byť použitý v rôznych situáciách
- vzor sa často spolieha charakteristiky objektu ako dedičnosť a polymorfizmus

Generatívne programovanie

- zahŕňa znovupoužitie štandardných vzorov a algoritmov, ktoré sú včlenené v generátore a parametrizované užívateľom. Potom je program automaticky generovaný
- veľmi cenovo výhodné, ale jeho aplikovateľnosť je značne obmedzená, pretože je relatívne malý počet aplikačných domén
- pre koncového užívateľa je to jednoduchšia metóda vývoja

Key points

- Výhody opäťovného použitia sú : nižšie náklady, rýchlejší vývoj softvéru a nižšie riziká
- Návrhové vzory sú na vysokej úrovni abstrakcie ktoré dokumentujú úspešné konštrukčné riešenie
- Programové generátory sú tiež zaobrajú softvériovým znovupoužitím- opakovane použiteľné koncepty sú uložené v systéme generátora

Softverova reciklacia 2

Aplikacne ramce, struktura(frameworks)

Ramce(frameworks) su subsystemove konstrukcie skladajuce sa z kolekcii abstraktnych a konkretnych tried a rozhrania medzi nimi.

Subsystem je implementovany pridanim komponentov ,ktore vyplnaju v niektorych castiach dizajnu a konkretizaciu abstraktnych tried v ramcoch.

Ramce su stredne velke subsystem ktore mozu byt znova pouzive (reused).

Framework classes

Systemova infrastruktura ramcov - Podporovať rozvoj systémových infraštruktúr ako sú komunikácie, užívateľské rozhrania akompilátory.

Middleware integracia ramcov - Standarty a triedy, ktoré podporujú komponent komunikácie a výmeny informácií.

Enterprise aplikacnych ramcov - Podporovať rozvoj špecifických typov aplikácie, ako sú telekomunikácie alebo finančné systémy.

Rozsirene ramce

Ramce sú všeobecné a sú rozšírené o vytvorenie konkrétnejších aplikácií alebo sub-systémov.

Rozsireny ramec zahrna –

- Pridanie konkrétnie triedy, ktoré dedia od operácie abstraktné triedy v rámci;
- Pridanie metód, ktoré sú voláne v reakcii na udalosti ktoré sú rozpoznane rámcom.

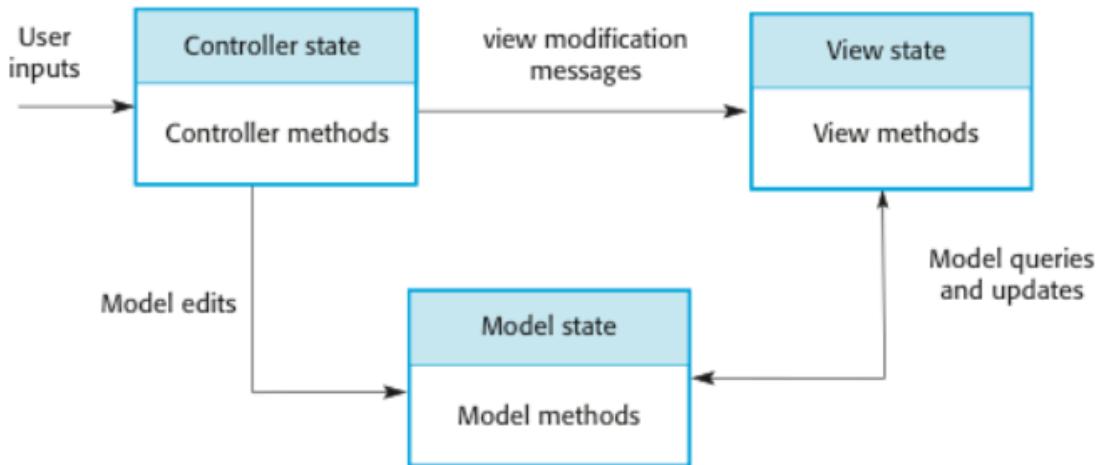
Problém s ramcami je ich zložitosť, ktorá znamená to, že trvá dlho, než ich budeme efektívne používať.

Model-view controller

Ramec systemovej infrastruktury pre GUI.

Umožňuje viacnásobné prezentácie objektu a samostatne interakcie s tymito prezentaciami.

MVC framework zahŕňa konkretizáciu počtu vzorov (ako je popísané vyššie under concept reuse).



Reuse aplikacny system

Zahŕňa opäťovné použitie celých aplikačných systémov buď konfiguráciu systému pre prostredie, alebo tým, že spája dve alebo viac systémov pre vytvorenie novej aplikácie.

dva prístupy sa zahrnujúce:

- Integráciu COTS produktov;
- Vývoj Produktovej súriny.

COTS produkt reuse

COTS - Commercial Off-The-Shelf systems.

COTS systémy sú zvyčajne kompletné aplikačné systémy, ktoré ponúkajú API.

- budovanie rozsiahlych systémov vďaka integrácii COTS systémov je teraz zivotaschopna stratégia rozvoja niektorých typov systému, ako sú napríklad e-commerce systémy.
- Hlavnou výhodou je rýchlejší vývoj aplikácií a, zvyčajne, nižšie náklady na vývoj.

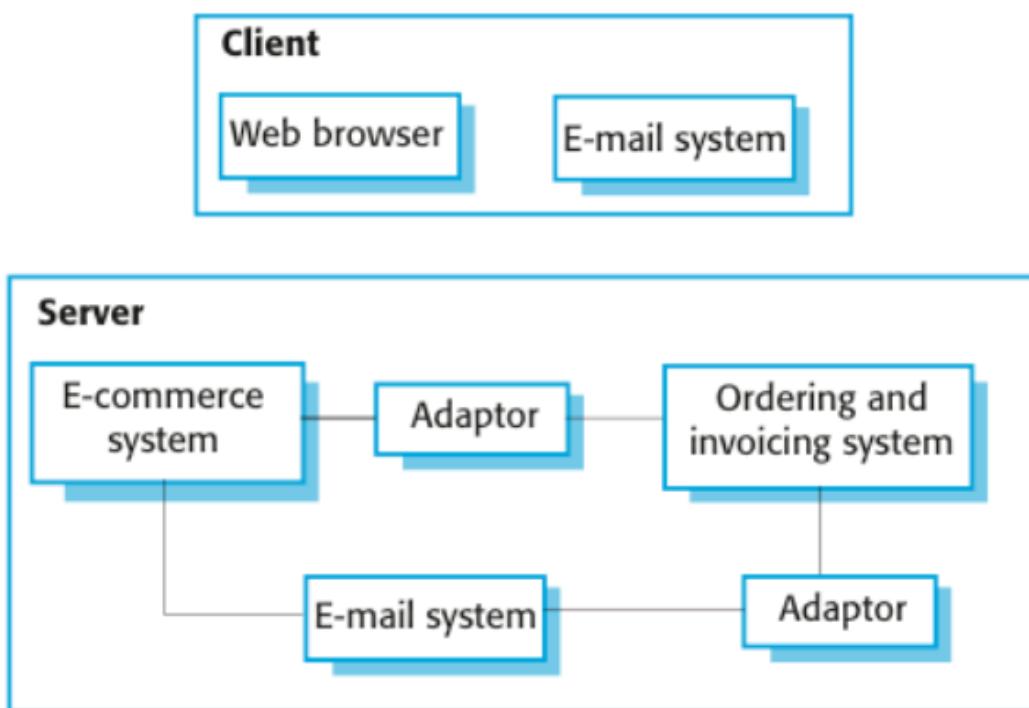
COTS design možnosti

Ktoré COTS produkty ponúkajú najvhodnejšie funkcionality?

Môže existovať niekoľko podobných produktov, ktoré môžu byť použité.

- Ako budú dátá vymienané ?
 - Jednotlivé produkty používajú svoje vlastné dátové štruktúry a formáty.
- Aké vlastnosti výrobku budu skutočne pouzívané?
 - Väčšina výrobcov má viac funkcií, ako je potreba.

Mali by ste sa pokúsiť odmietnut prístup k nevyužívanym funkciam.



COTS produkty reused

Na strane klienta sa používajú štandardné e-mailové a webové prehliadacie programy.

Na serveri, e-commerce platforma má byť integrovaná s existujúcim objednávkovým systémom.

- To zahrňa pisanie adaptora („an adaptor“ wtf?) tak, že si môžu vymieňať dátu.
- To tiež vyžaduje adaptora pre príjem dát z objednávania a fakturácneho systému.

Problémy COTS systémovej integrácie

Nedostatok kontroly nad funkcionalitou a výkonom

- COTS systémy môžu byť menej účinné, než sa zdá

Problémy s COTS systémovou interoperabilitou(súčinnosťou ?)

- Rôzne COTS systémy môžu robiť rôzne predpoklady, to znamená, že integrácia je obtiažná

Ziadna kontrola nad vývojom systému

- COTS predajcovia nie sú užívatelia systému, ktorí ho vývyjajú

Podpora od COTS predajcov

- COTS predajcovia nemusia ponúkať podporu po celú dobu životnosti výrobku

Software product lines

Software product lines alebo rodiny aplikácií sú aplikácie s vseobecnymi funkciami, ktoré môžu byť prispôsobené a konfigurované pre použitie v konkrétnom kontexte.

Prispôsobenie môže zahŕňať:

- Komponentný a konfiguráciu systému;
- Pridanie nové komponenty do systému;
- Výber z knižnice existujúcich komponentov;
- Úprava komponentov pre splnenie nových požiadaviek.

COTS produktova specializacia

Platforma špecializácie

- Rôzne verzie aplikácie sú vyvinuté pre rôzne platformy.

Environment specializacia

- Rôzne verzie aplikácie sú vytvorené zvládnuť rôzne operačné prostredie, napr rôzne typy komunikačných zariadení.

Functional specializacia

- Rôzne verzie aplikácie sú vytvorené pre zákazníkov s rôznymi požiadavkami.

Proces špecializácie

- Rôzne verzie aplikácie sú vytvorené pre podporu rôznych obchodných procesov.

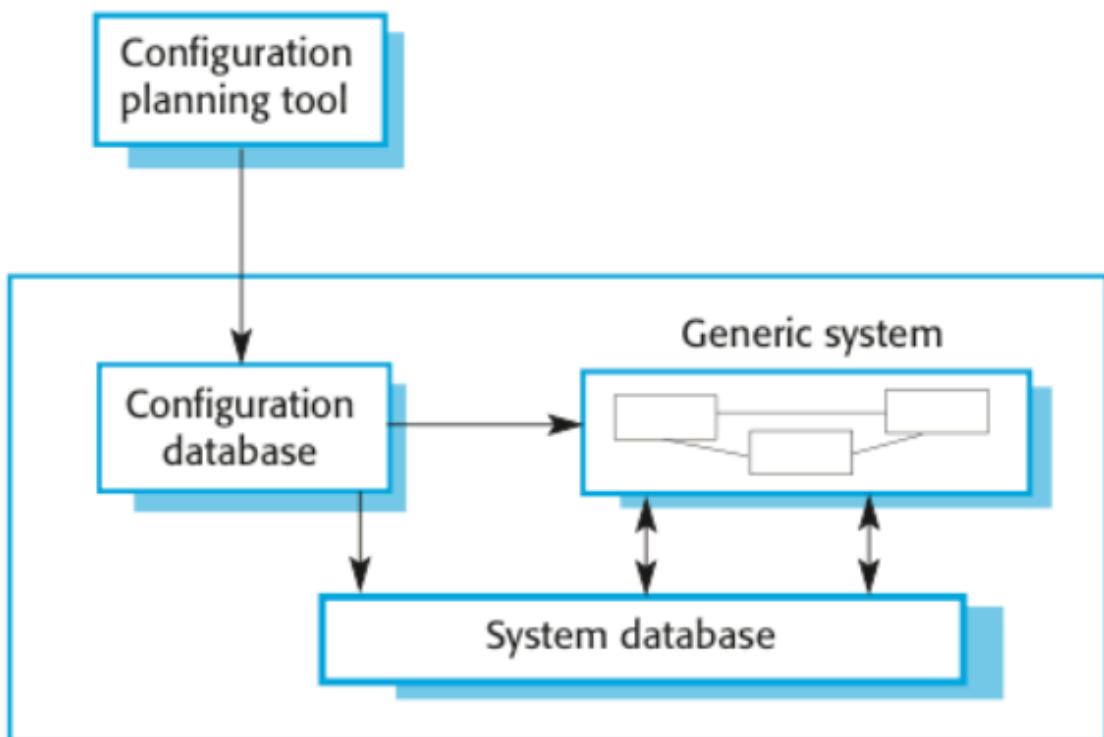
COTS konfiguracia

konfigurácia doby nasadenia (deployment time)

- všeobecné(generic) systém je nakonfigurovaný vložením znalostí požiadaviek zákazníka a obchodných procesov. Softvér sám o sebe sa nezmení.

Design time konfiguracia

- obyčajný generic kód je upravený a zmenený podľa požiadaviek jednotlivých zákazníkov.



ERP systémy

Enterprise Resource Planning (ERP) systém je generic systém, ktorý podporuje bežne obchodné procesy, ako je objednavanie a fakturácia, výroba, apod.

Sú široko používané vo veľkých spoločnostiach

- Predstavujú pravdepodobne najčastejšu formu opäťovného využitia softvéru.

Generic jadro je upravene tým, že zahŕňa moduly a začlenenuje vedomosti obchodných procesov a pravidiel.

Design time konfiguracia

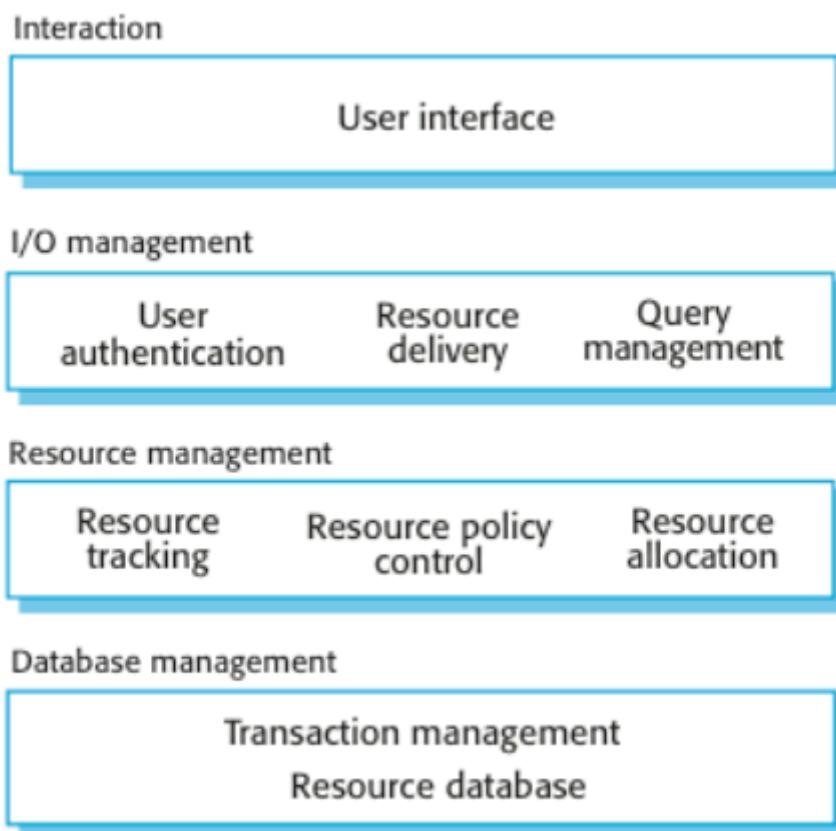
Software product lines, ktoré sú konfigurovane v čase „design time“ sú konkretizované generic aplikacnymi architektúrami.

Generic produkt sa zvyčajne objaví po skúsenostiach s konkrétnymi produktmi.

Product line architectures

Architektúra musí byť štruktúrovaná tak, aby oddelila jednotlivé podsystémy a umožniť im byť upravená.

Architektura by mala separovať entity a ich popisy a vyššej úrovne v prístupe k systému prostredníctvom entít, popisov a nie priamo.



Sprava vozidiel (vehicle despatching)

špecializovaný systém pre správu zdrojov, kde cieľom je pridelit' prostriedky(vehicles) pre spracovanie udalosti.

Úpravy zahŕňajú:

- Na úrovni užívateľského rozhrania, tam sú komponenty pre displej obsluhy a komunikácie
- Na úrovni riadenia I / O, tam sú komponenty, ktoré spracúvajú overovanie, podávanie správ a plánovanie trasy;
- Na úrovni riadenia zdrojov, tam sú komponenty pre umiestnenie vozidiel a odoslania(despatch), riadenie stavu vozidla a logovanie incidentov;
- Databáza obsahuje databázy zariadení, vozidlá a mapy.

Key points

Aplikačné rámce sú kolekcie konkrétnych a abstraktných objektov, ktoré sú určené pre opakované použitie prostredníctvom špecializácie.

Opäťovné použitie výrobku COTS sa zaobrá opäťovným použitím vo velkom, off-the-shelf systems.

K Problému s opäťovným použitím COTS patrí nedostatok kontroly nad funkcionálou, výkonom a vývojom a problémy s inter-operation.(súčinnostou ?)

ERP systemy sú vytvorené konfiguraciou generic systemu s informáciami o podnikaní zákazníka.

Software product lines sú súvisiace aplikácie vyvinuté okolo spoločného jadra zdielanej funkcionality.

Security Engineering

Nástroje, techniky a metódy na podporu vývoja a údržby systému odolávajúcich škodlivým útokom, ktorých cieľom je poškodiť systém alebo jeho dátu. Súčasť oblasti počítačovej bezpečnosti.

Application/Infrastructure security

Zabezpečenie aplikácií je problém softvérového inžinierstva, kde je systém navrhnutý na odolávanie útokom. Zabezpečenie infraštruktúry je problémom správy systému, kde infraštruktúra je nakonfigurovaná odolávať útokom. Predmetom tejto kapitoly je zabezpečenie aplikácií.

Základné pojmy

<u>Asset</u> (majetok)	hodnotný systémový zdroj, ktorý treba chrániť	jednotlivé záznamy pacientov
<u>Exposure</u> (expozícia)	možná strata alebo poškodenie z úspešného útoku, môže sa jednať o stredu údajov alebo času a úsilia potrebného na opravu	strata financií, kvôli strate pacientov, ktorí nedôverujú systému
<u>Vulnerability</u> (zraniteľnosť)	slabina počítačového systému, ktorá môže byť využitá na spôsobenie škôd	slabé zabezpečenie hesiel, rovnaké ID pacienta ako jeho meno
<u>Attack</u> (útok)	využitie zraniteľnosti systému, väčšinou zvonku, úmyslom je poškodenie	zosobnenie (impersonation) oprávneného užívateľa
<u>Threats</u> (hrozby)	okolnosti, ktoré môžu spôsobiť stratu alebo poškodenie, zraniteľnosť, ktorá je subjektom útoku	získanie kontroly neoprávneného užívateľa hádaním prihlásovacích údajov autorizovaného užívateľa
<u>Control</u> (kontrola)	ochranné opatrenia, ktoré znížujú zraniteľnosť systému, napríklad šifrovanie	kontrola hesiel, aby neobsahovali slová zo slovníkov

Security controls (bezpečnostné kontroly)

Kontroly určené na zabezpečenie, že útoky nebudú úspešné (fault avoidance). Kontroly určené na detekciu a odrazenie útokov (fault detection & tolerance). Kontroly určené na podporu obnovenia z problémov (fault recovery).

Bezpečnostné požiadavky

- Informácie o pacientovi musí byť stiahnuté na začiatok a kliniky reláciu zabezpečenej oblasti na klientský systém, ktorý sa používa nemocničný personál.
- Informácie o pacientovi nesmie byť udržiavané na klientov systému po klinike zasadnutie ukončil.
- A záznam na samostatnom počítači z databázy

server musí byť zachovaná všetkých vykonaných zmien do databázy systému.

- Patient information must be downloaded at the start of a clinic session to a secure area on the system client that is used by clinical staff.
- Patient information must not be maintained on system clients after a clinic session has finished.
- A log on a separate computer from the database server must be maintained of all changes made to the system database.

Posúdenie rizík životného cyklu

Hodnotenie počas vývoja systému aj po nasadení. Dostupných je viacero informácií - systémová platforma, organizácia dát, middleware a systémová architektúra. Preto môžu byť odhalené chyby, ktoré vznikli pri návrhu.

Príklady rozhodnutí pri návrhu

Overenie autentickosti užívateľa pomocou kombinácie meno/heslo. Systémová architektúra je klient-server s prístupom užívateľov pomocou bežného prehliadača. Informácie sú prezentované ako editovateľná web forma.

Návrh pre bezpečnosť

Návrh architektúry - ako rozhodnutie o architektúre ovplyvní bezpečnosť systému?

Vhodné postupy - aké sú vhodné postupy pri návrhu bezpečného systému?

Návrh nasadenia - ako podporiť systém aby sa predišlo zavedeniu chýb pri nasadení do použitia?

Návrh architektúry

Ochrana - ako má byť systém organizovaný, aby bol dôležitý majetok chránený pred vonkajšími útokmi?

Distribúcia - ako sa má majetok distribuovať, aby sa minimalizovala účinnosť útokov?

Potenciálne konflikty - ak je majetok distribuovaný, je drahší na zabezpečenie

Ochrana

Ochrana na úrovni platformy, aplikácie alebo záznamov.

Platform level protection

System authentication

System authorization

File integrity management

Application level protection

Database login

Database authorization

Transaction management

Database recovery

Record level protection

Record access authorization

Record encryption

Record integrity management

Usmernenia návrhu

Zapúzdrenie vhodných postupov návrhu bezpečného systému slúžiace dvom účelom:

- zvýšenie povedomia o bezpečnostných problémoch v tíme softvérových inžinierov
- môžu byť použité ako základ revízneho zoznamu použitého pri validácii systému

Usmerneina: Základné rozhodnutia na explicitnej bezpečnostnej politike. Vyhnut' sa zlyhaniu v jednom bode. Zlyhať bezpečne. Vyrovnať použiteľnosť a bezpečnosť. Byť si vedomí možností sociálneho inžinierstva. Využiť redundanciu a rozmanitosť na eliminovanie rizík. Overovať všetky vstupy. Rozčleniť svoj majetok. Návrh pre nasadenie. Návrh pre obnoviteľnosť.

KeyPoints:

Bezpečnostné inžinierstvo sa zaoberá tým, ako vyvíjať systémy, ktoré môžu odolávať škodlivým útokom

- Bezpečnostné hrozby môže byť ohrozením dôvernosti, integrity alebo dostupnosť systému alebo jeho dát
- Design pre zabezpečenie zahrňa architektonické riešenie, nasledujúce dobrú projekčnú prax a minimalizujúc zavádzanie slabých miest systému
- Kľúčové problémy pri navrhovaní bezpečnej architektúry sú organizovanie štruktúry na ochranu majetku a distribúciu aktív na minimalizáciu strát
- Všeobecné bezpečnostné pokyny zvyšujú citlosť návrhárov pri riešení bezpečnostných otázok a slúžia ako hodnotiace zoznamy

Security Development Lifecycle (SDL)

Je proces vývoja softvéru, ktorý pomáha vývojárom vytvárať lepšie zabezpečený softvér a zabezpečiť všetky potrebné požiadavky na vývoj a zároveň znížiť celkové náklady.

Training Phase

1. Základný bezpečnostný tréning: toto cvičenie je predpokladom pre vykonávanie SDL. Koncepcia budovania lepšieho softvéru zahŕňa bezpečnostný design, bezpečné kódovanie, testovanie zabezpečenia a osvedčených postupov okolitého súkromia (inak som to nevedel preložiť).

Requirements Phase

2. Stanoviť zabezpečenie a ochranu osobných údajov: Definovanie a integrácia bezpečnosti a ochrany osobných údajov pomáha, aby bolo ľahšie identifikovať kľúčové míľníky a výsledky a tým pádom minimalizovali narušenie plánov a harmonogramov.

3. Vytvoriť kvalitné Gates / Bug Bary: Vymedzenie minimálnej priateľnej úrovne bezpečnosti a kvality ochrany súkromia, na začiatku tým pomáha pochopíť riziká spojené s otázkami bezpečnosti, identifikovať a opraviť bezpečnostné chyby počas vývoja a uplatňovať štandard v rámci celého projektu.

4. Vykonáť zabezpečenia a ochranu súkromných ustanovení rizík: Skúmanie dizajnu softvéru na základe nákladov a regulačných požiadaviek, pomáha tímu identifikovať, ktoré časti projektu budú vyžadovať threat modelling a bezpečnosť dizajnu softvéru pred spustením a určiť súkromný dopad hodnotenia produktu alebo služby.

Design Phase

5. Zaviesť dizajnové požiadavky: Pomocou zabezpečenia a ochrany osobných údajov pomáhamo minimalizovať riziko narušenia harmonogramu a znížiť náklady na projekt.

6. Attack Surface Analysis/Reduction: Znížiť príležitosť na objavenie slabiny a zraniteľnosti pre potenciálnych útočníkov vyžaduje dôkladne analyzovať a obmedziť prístup k systémovým službám, uplatňovať princíp posledného privilégia a využiť obranné vrstvy, tam kde sú.

7. Použite Threat Modeling: Použiť štruktúrovaný prístup k scenáru hrozieb pri navrhovaní, pomáha tímu efektívnejšie a lacnejšie identifikovať slabé miesta, určenie rizík vyplývajúcich z tohto hrozieb a stanoviť vhodnú závažnosť rizika.

Implementation Phase

8. Použiť schválené nástroje: Publikovanie zoznamu schválených nástrojov a súvisiacich bezpečnostných kontrol pomáha automatizovať a presadzovať bezpečnostné postupy, ľahko pri nízkych nákladoch. Viesť pravidelne aktualizovaný zoznam znamená, že sú použité najnovšie nástroje a umožňuje tým integráciu nových funkcií a ochranu bezpečnosti.

9. Kritizovať nebezpečné funkcie: Analyzovať všetky funkcie a rozhrania API projektu a zákaz tých, ktoré sú určené ako nebezpečné pomáha znižovať potenciálne bezpečnostné chyby s veľmi malými nákladmi. Konkrétnie opatrenia zahŕňajú použitie hlavičkových súborov, novšie komplilátory, alebo skenovanie pre kontrolu kódu, a potom i nahradenia ho bezpečnejšími alternatívami

10. Vykonanie statickej analýzy: Analýza zdrojového kódu nám poskytne škálovateľný spôsob zabezpečenia a pomáha zaistiť , aby bolo dodržiavaná bezpečná politika kódovania.

Verification Phase

11. Vykonanie dynamickej analýzy: Vykonanie funkcie kontroly overovania softvéru runtime pomocou nástrojov, ktoré sledujú správanie aplikácie pre poškodenie pamäti, otázok používateľov s oprávneniami, a iných kritických bezpečnostných problémov.

12. Fuzz Testovanie: Zlyhanie programu vyvolané, zámerným poškodenými alebo náhodnými dátami pomáha odhaliť možné problémy zabezpečenia pred spustením.

13. Attack Surface Review: Kontrola týmto útokom pomáha zabezpečiť , aby boli prijaté akékoľvek konštrukčné a a vykonávacie zmeny aplikácie do systému a nové vektory útoku vytvorené v dôsledku zmien vrátane modelov služieb.

Release Phase

14. vytvorenie Incident Response Plan: Príprava plánu pre Incident Response má zásadný význam pre pomoc pri riešení nových hrozieb , ktoré sa môžu objaviť v priebehu času . To zahŕňa identifikáciu vhodných bezpečnostných mimoriadnych kontaktov a zaviesť bezpečnostné plány pre kód .

15. Conduct Final Security Review: Zámerne preskúmanie všetkých bezpečnostných činností , ktoré boli vykonané pomáha zaistiť pripravenosť softvéru. Záverečná Security Review (FSR) zvyčajne zahŕňa výberový model hrozieb , nástroje výstupov a výkon proti kvalitným brány a chýb barov definovaných v požiadavkách fáze(Requirement Phrase).

16. Certifikácia a vydanie archívu: Certifikačný program pred uvoľnením pomáha zaistiť , aby boli splnené požiadavky zabezpečenia a ochrany osobných údajov. Archivácia všetkých

údajov je potrebná pre vykonávanie po uvoľnení servisných úloh a pomáha znížiť dlhodobé náklady spojené s softvérovým inžinierstvom.

Response Phase

17.Prevedenie Incident Response Plan: Byť schopný realizovať Response plán Incident začatý vo fáze Release je nevyhnutné k pomoci a ochrane zákazníkov pred nebezpečenstvom alebo zraniteľnosti ochrany osobných údajov, ktoré sa objavujú.