



Kapitola 1- Úvod

Softvérové inžinierstvo



- ✧ Ekonomiky VŠETKÝCH vyspelých krajín sú závislé od softvéru.
- ✧ Stále viac systémov je riadených softvérom
- ✧ **Softvérové inžinierstvo** sa zaoberá teóriami, metódami a nástrojmi pre profesionálny vývoj softvéru.
- ✧ Výdavky na softvér predstavujú významnú časť HNP vo všetkých rozvinutých krajinách.



Náklady na softvér

- ✧ Náklady na softvér často dominujú v nákladoch na počítačový systém. Náklady na softvér na PC sú často vyššie ako náklady na hardvér.
- ✧ Údržba softvéru je drahšia ako jeho vývoj. Pri systémoch s dlhou životnosťou môžu náklady na údržbu niekoľkonásobne prevýšiť náklady na vývoj.
- ✧ **Softvérové inžinierstvo** sa zaoberá nákladovo-efektívnym vývojom softvéru.

Zlyhanie softvérového projektu



✧ Zvyšovanie zložitosti systému

- Keďže nové techniky softvérového inžinierstva nám pomáhajú budovať väčšie a komplexnejšie systémy, požiadavky sa menia. Systémy musia byť postavené a dodávané rýchlejšie; sú potrebné väčšie, ešte zložitejšie systémy; systémy musia mať nové schopnosti, ktoré sa predtým považovali za nemožné.

✧ Nepoužívanie metód softvérového inžinierstva

- Je pomerne jednoduché písat' počítačové programy bez použitia metód a techník softvérového inžinierstva. Mnoho spoločností sa vrhlo na vývoj softvéru, pretože ich produkty a služby sa vyvýiali. Vo svojej každodennej práci nepoužívajú metódy softvérového inžinierstva. V dôsledku toho je ich softvér často drahší a menej spoľahlivý, ako by mal byť.

Často kladené otázky o softvérovom inžinierstve



Otázka	Odpoveď
Čo je softvér?	Počítačové programy a súvisiaca dokumentácia. Softvérové produkty môžu byť vyvinuté pre konkrétneho zákazníka alebo môžu byť vyvinuté pre všeobecný trh.
Aké sú vlastnosti dobrého softvéru?	Dobrý softvér by mal používateľovi poskytovať požadovanú funkčnosť a výkon a mal by byť udržiavateľný, spoľahlivý a použiteľný.
Čo je softvérové inžinierstvo ?	Softvérové inžinierstvo je inžinierska disciplína, ktorá sa zaoberá všetkými aspektmi výroby softvéru.
Aké sú základné činnosti softvérového inžinierstva ?	Specifikácia softvéru, vývoj softvéru, validácia softvéru a ďalší vývoj softvéru.
Aký je rozdiel medzi softvérovým inžinierstvom a počítačovou vedou?	Informatika sa zameriava na teóriu a základy; softvérové inžinierstvo sa zaoberá praktickými aspektmi vývoja a dodávania užitočného softvéru.
Aký je rozdiel medzi softvérovým inžinierstvom a systémovým inžinierstvom?	Systémové inžinierstvo sa zaoberá všetkými aspektmi vývoja počítačových systémov vrátane hardvéru, softvéru a procesného inžinierstva. Softvérové inžinierstvo je súčasťou tohto všeobecnejšieho procesu.

Často kladené otázky o softvérovom inžinierstve (2)



Otázka	Odpoveď
Aké sú hlavné výzvy, ktorým čelí softvérové inžinierstvo?	Vyrovnanie sa s rastúcou rozmanitosťou, požiadavkami na skrátenie dodacích lehôt a vývoj dôveryhodného softvéru.
Aké sú náklady na softvérové inžinierstvo?	Zhruba 60 % nákladov na softvér tvoria náklady na vývoj, 40 % tvoria náklady na testovanie. V prípade zákazkového softvéru náklady na ďalší vývoj (evolúciu) často prevyšujú náklady na vývoj.
Aké sú najlepšie techniky a metódy softvérového inžinierstva?	Zatiaľ čo všetky softvérové projekty musia byť profesionálne riadené a vyvíjané, pre rôzne typy systémov sú vhodné rôzne techniky. Napríklad hry by sa mali vždy vyvíjať s použitím série prototypov, zatiaľ čo bezpečnostné kritické riadiace systémy vyžadujú vypracovanie úplnej a analyzovateľnej špecifikácie. Nedá sa teda povedať, že jedna metóda je lepšia ako druhá.
Aké rozdiely priniesol web od softvérového inžinierstva?	Web viedol k dostupnosti softvérových služieb a možnosti vývoja vysoko distribuovaných systémov založených na službách. Vývoj webových systémov viedol k významnému pokroku v programovacích jazykoch a opäťovnom použití softvéru.

Softvérové produkty



✧ Generické produkty

- Samostatné systémy, ktoré sú predávané a predávané každému zákazníkovi, ktorý si ich želá kúpiť.
- Specifikáciu toho, čo by mal softvér robiť, vlastní vývojár softvéru a rozhodnutia o zmene softvéru robí vývojár.

✧ Prispôsobené produkty

- Softvér, ktorý je objednaný konkrétnym zákazníkom, aby vyhovoval jeho vlastným potrebám.
- Specifikáciu toho, čo by mal softvér robiť, vlastní zákazník pre softvér a rozhoduje o zmenách softvéru, ktoré sú potrebné.

Základné atribúty dobrého softvéru



Charakteristika produktu	Popis
Udržiavateľnosť	Softvér by mal byť napísaný tak, aby sa mohol vyuvíjať, aby vyhovoval meniacim sa potrebám zákazníkov. Toto je kritický atribút, pretože zmena softvéru je nevyhnutnou požiadavkou meniaceho sa obchodného prostredia.
Spoľahlivosť a bezpečnosť	Spoľahlivosť softvéru zahŕňa celý rad charakteristík vrátane spoľahlivosti, zabezpečenia a bezpečnosti. Spoľahlivý softvér by nemal spôsobiť fyzické alebo ekonomicke škody v prípade zlyhania systému. Používatelia so zlými úmyslami by nemali mať prístup k systému alebo ho poškodiť.
Efektívnosť	Softvér by nemal zbytočne využívať systémové prostriedky, ako sú pamäť a cykly procesora. Efektívnosť teda zahŕňa odozvu, čas spracovania, využitie pamäte atď.
Prijateľnosť	Softvér musí byť prijateľný pre typ používateľov, pre ktorých je určený. To znamená, že musí byť zrozumiteľný, použiteľný a kompatibilný s inými systémami, ktoré používajú.

Ďalšie atribúty dobrého softvéru



Bezpečnosť	Zrozumiteľnosť	Prenosnosť
Bezpečnosť	Testovateľnosť	Použiteľnosť
Spoľahlivosť	Prispôsobivosť	Opätná použiteľnosť
Odolnosť	Modularita	Efektívnosť
Robustnosť	Zložitosť	Učenosť

Vhodnosť softvéru na daný účel



- ✧ Bol softvér riadne otestovaný?
- ✧ Je softvér dostatočne spoločalivý na to, aby sa dal použiť?
- ✧ Je výkon softvéru prijateľný pre bežné použitie?
- ✧ Je softvér použiteľný?
- ✧ Je softvér dobre štruktúrovaný a zrozumiteľný?
- ✧ Boli v procese vývoja dodržané programovacie a dokumentačné štandardy?

Softvérové inžinierstvo



- ✧ **Softvérové inžinierstvo** je inžinierska disciplína, ktorá sa zaoberá všetkými aspektmi výroby softvéru od počiatočných štádií špecifikácie systému až po údržbu systému po jeho uvedení do prevádzky.
- ✧ **Inžinierska disciplína**
 - Používanie vhodných teórií a metód na riešenie problémov s ohľadom na organizačné a finančné obmedzenia.
- ✧ **Všetky aspekty výroby softvéru**
 - Nielen technický proces vývoja. Taktiež projektový manažment a vývoj nástrojov, metód atď. na podporu produkcie softvéru.

Význam softvérrového inžinierstva



- ✧ Jednotlivci a spoločnosť sa čoraz viac spoliehajú na pokročilé softvérrové systémy. Musíme byť schopní vyrábať spoľahlivé a dôveryhodné systémy hospodárne a rýchlo.
- ✧ Z dlhodobého hľadiska je zvyčajne lacnejšie používať metódy a techniky softvérrového inžinierstva pre softvérrové systémy, než len písat programy, ako keby to bol projekt osobného programovania. Pre väčšinu typov systémov tvoria väčšinu nákladov náklady na zmenu softvéru po jeho uvedení do prevádzky.

Softvérové procesné činnosti



- ✧ **Špecifikácia softvéru**, kde zákazníci a inžinieri definujú softvér (požiadavky), ktorý sa má vyrábať, a obmedzenia jeho prevádzky.
- ✧ **Vývoj softvéru**, kde je softvér navrhnutý a naprogramovaný.
- ✧ **Validácia softvéru**, kde sa kontroluje softvér, aby sa zabezpečilo, že je to, čo zákazník vyžaduje.
- ✧ **Evolúcia softvéru**, kde sa softvér upravuje tak, aby odrážal meniace sa požiadavky zákazníkov a trhu.

Všeobecné problémy, ktoré ovplyvňujú softvér



✧ Heterogenita

- V čoraz väčšej miere sa vyžaduje, aby systémy fungovali ako distribuované systémy v sietiach, ktoré zahŕňajú rôzne typy počítačov a mobilných zariadení.

✧ Obchodné a spoločenské zmeny

- Podnikanie a spoločnosť sa neuveriteľne rýchlo menia, pretože rozvíjajúce sa ekonomiky sa rozvíjajú a sú k dispozícii nové technológie. Musia byť schopní zmeniť svoj existujúci softvér a rýchlo vyvinúť nový softvér.

Všeobecné problémy, ktoré ovplyvňujú softvér (2)



✧ Bezpečnosť a dôvera

- Kedže softvér je prepojený so všetkými aspektmi nášho života, je nevyhnutné, aby sme mu mohli dôverovať.

✧ Rozsah

- Softvér sa musí vyvíjať vo veľmi širokom rozsahu, od veľmi malých vstavaných systémov v prenosných alebo nositeľných zariadeniach až po internetové cloudové systémy, ktoré slúžia globálnej komunite.

Rozmanitosť softvérového inžinierstva



- ✧ Existuje mnoho rôznych typov softvérových systémov a neexistuje univerzálna množina softvérových techník, ktoré by sa dal použiť na všetky tieto systémy.
- ✧ Použité metódy a nástroje softvérového inžinierstva závisia od typu vyvíjanej aplikácie, požiadaviek zákazníka a skúseností vývojového tímu.

Typy aplikácií



✧ Samostatné aplikácie

- Ide o aplikačné systémy, ktoré bežia na lokálnom počítači, ako je PC. Zahŕňajú všetky potrebné funkcie a nemusia byť pripojené k sieti.

✧ Interaktívne aplikácie založené na transakciách

- Aplikácie, ktoré sa spúšťajú na vzdialenom počítači a používatelia k nim pristupujú z ich vlastných počítačov alebo terminálov. Patria sem webové aplikácie, ako sú aplikácie elektronického obchodu.

✧ Vstavané riadiace systémy

- Ide o softvérové riadiace systémy, ktoré riadia a riadia hardvérové zariadenia. Číselne existuje pravdepodobne viac vstavaných systémov ako akýkoľvek iný typ systému.



Typy aplikácií (2)

✧ Systémy dávkového spracovania

- Ide o obchodné systémy, ktoré sú určené na spracovanie údajov vo veľkých dávkach. Spracujú veľké množstvo jednotlivých vstupov, aby vytvorili zodpovedajúce výstupy.

✧ Zábavné systémy

- Ide o systémy, ktoré sú primárne určené na osobné použitie a ktorých cieľom je zabaviť používateľa.

✧ Systémy pre modelovanie a simuláciu

- Sú to systémy vyvinuté vedcami a inžiniermi na modelovanie fyzikálnych procesov alebo situácií, ktoré zahŕňajú mnoho samostatných interagujúcich objektov.

Typy aplikácií (3)



✧ Systémy zberu údajov

- Sú to systémy, ktoré zhromažďujú údaje zo svojho prostredia pomocou sady senzorov a odosielajú tieto údaje do iných systémov na spracovanie.

✧ Systémy systémov

- Ide o systémy, ktoré sú zložené z množstva iných softvérových systémov.

Základy softvérového inžinierstva



- ✧ Niektoré základné princípy platia pre všetky typy softvérových systémov, bez ohľadu na použité techniky vývoja:
 - Systémy by sa mali vyvíjať pomocou riadeného a pochopeného vývojového procesu. Samozrejme, pre rôzne typy softvéru sa používajú rôzne procesy.
 - Spoločalivosť a výkon sú dôležité pre všetky typy systémov.
 - Dôležité je pochopenie a správa softvérových špecifikácií a požiadaviek (čo by mal softvér robiť).
 - Ak je to vhodné, mali by ste radšej použiť softvér, ktorý už bol vyvinutý, ako písat nový softvér.



Kapitola 10 – Spořahlivé systémy

Spoľahlivosť systému



- ✧ Pre mnoho počítačových systémov je najdôležitejšou vlastnosťou systému spoľahlivosť systému.
- ✧ **Spoľahlivosť systému odráža mieru dôvery používateľa v tento systém.** Odráža mieru dôvery používateľa, že bude fungovať tak, ako používatelia očakávajú, a že pri bežnom používaní „nezlyhá“.
- ✧ Spoľahlivosť pokrýva súvisiace atribúty systémov ako spoľahlivosť, dostupnosť a bezpečnosť. Všetky sú vzájomne závislé.

Dôležitosť spoľahlivosti



- ✧ Zlyhania systému môžu mať rozsiahle následky s veľkým počtom ľudí postihnutých zlyhaním.
- ✧ Systémy, ktoré nie sú spoľahlivé a sú nespoľahlivé, nebezpečné alebo neisté, môžu ich používatelia odmietnuť.
- ✧ Náklady na zlyhanie systému môžu byť veľmi vysoké, ak zlyhanie vedie k ekonomickým stratám alebo fyzickým škodám.
- ✧ Nespoľahlivé systémy môžu spôsobiť stratu informácií s vysokými následnými nákladmi na obnovu.

Príčiny zlyhania



✧ Porucha hardvéru

- Hardvér zlyhá v dôsledku konštrukčných a výrobných chýb alebo preto, že komponenty dosiahli koniec svojej prirodzenej životnosti.

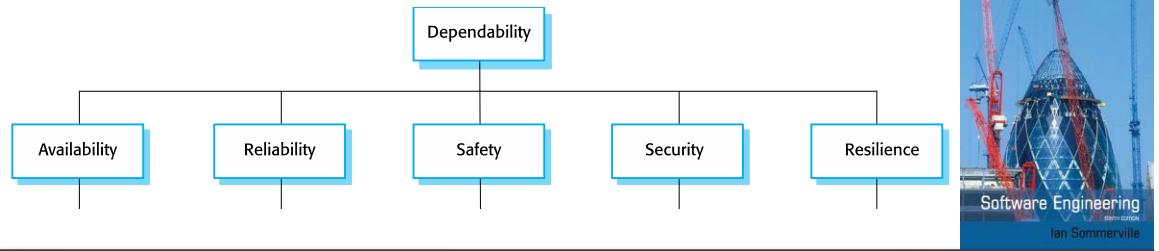
✧ Zlyhanie softvéru

- Softvér zlyhá v dôsledku chýb v jeho špecifikácii, návrhu alebo implementácii.

✧ Prevádzková porucha

- Ľudskí operátori robia chyby. Možno najväčšia príčina zlyhania systému v sociálno-technických systémoch.

Hlavné vlastnosti



- ✧ **Dostupnosť** - súčasť informačnej bezpečnosti
 - Pravdepodobnosť, že systém bude v prevádzke a bude schopný poskytovať používateľom užitočné služby.
- ✧ **Výpočtová spoľahlivosť**
 - Pravdepodobnosť, že systém bude správne poskytovať služby podľa očakávania používateľov.
- ✧ **Ochrana zdravia, života a prostredia**
 - Posúdenie toho, aká je pravdepodobnosť, že systém spôsobí škody ľuďom alebo životnému prostrediu.
- ✧ **Informačná bezpečnosť**
 - Posúdenie toho, aká je pravdepodobnosť, že systém dokáže odolať náhodným alebo úmyselným prienikom. C-I-A: dôvernosť, integrita, **dostupnosť**.
- ✧ **Odolnosť**
 - Posúdenie toho, ako dobre môže systém zachovať kontinuitu svojich kritických služieb v prítomnosti rušivých udalostí, ako je zlyhanie zariadenia a kybernetické útoky.

Ďalšie vlastnosti spoľahlivosti



✧ Opraviteľnosť

- Odráža mieru, do akej je možné systém opraviť v prípade poruchy

✧ Udržiavateľnosť

- Odráža mieru, do akej je možné systém prispôsobiť novým požiadavkám;

✧ Tolerancia chýb

- Odráža mieru, do akej sa možno vyhnúť chybám pri zadávaní údajov a tolerovať ich.

Závislosti atribútov spoľahlivosti



- ✧ Bezpečná prevádzka systému závisí od dostupnosti a spoľahlivosti systému.
- ✧ Systém môže byť nespoľahlivý, pretože jeho údaje boli poškodené vonkajším útokom.
- ✧ Útoky odmietnutia služby na systém sú určené na to, aby bol nedostupný.
- ✧ Ak je systém infikovaný vírusom, nemôžete si byť istí jeho spoľahlivosťou alebo bezpečnosťou.



Typy kritických systémov

✧ Systémy kritické z hľadiska bezpečnosti

- Porucha má za následok stratu života, zranenie alebo poškodenie životného prostredia
- *Chemický systém ochrany rastlín*

✧ Systémy kritické pre danú úlohu alebo cieľ

- Zlyhanie má za následok zlyhanie nejakej cieľovo orientovanej činnosti
- *Navigačný systém kozmickej lode*

✧ Systémy kritické pre podnikanie

- Zlyhanie má za následok vysoké ekonomicke straty
- *Účtovný systém zákazníkov v banke*

Dosiahnutie spoľahlivosti



- ✧ Vyhnite sa náhodným chybám pri vývoji systému.
- ✧ Navrhnite V & V procesy, ktoré sú efektívne pri odhalovaní zvyškových chýb v systéme.
- ✧ Navrhnite systémy tak, aby bol tolerantný: aby systém mohol pokračovať v prevádzke, keď sa vyskytnú poruchy
- ✧ Navrhnite ochranné mechanizmy, ktoré chránia pred vonkajšími útokmi.

Dosiahnutie spoľahlivosti (2)



- ✧ Správne nakonfigurujte systém pre jeho operačné prostredie .
- ✧ Zahrňte systémové schopnosti na rozpoznanie a odolanie kybernetickým útokom.
- ✧ Zahrňte mechanizmy obnovy, ktoré pomôžu obnoviť normálnu systémovú službu po zlyhaní.

Ekonomika spoľahlivosti

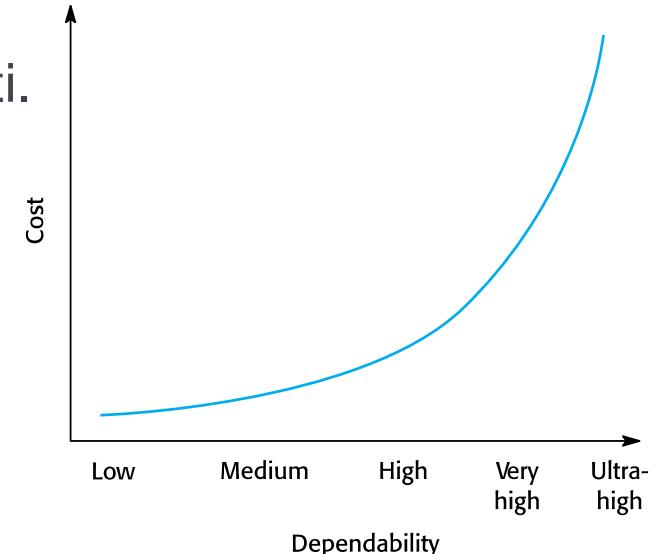


- ✧ Kvôli veľmi vysokým nákladom na dosiahnutie spoľahlivosti môže byť nákladovo efektívnejšie akceptovať nedôveryhodné systémy a zaplatiť náklady na zlyhanie
- ✧ To však závisí od sociálnych a politických faktorov. Povest' produktov, ktorým nemožno dôverovať, môže viest' ku strate budúceho podnikania
- ✧ Závisí od typu systému – najmä pre obchodné systémy môže byť primeraná mierna úroveň spoľahlivosti

Náklady na spoľahlivosť



- ✧ Náklady na spoľahlivosť majú tendenciu exponenciálne rásť, pretože je potrebná vyššia úroveň spoľahlivosti .
- ✧ Sú na to dva dôvody
 - Použitie **drahších vývojových techník** a hardvéru, ktoré sú potrebné na dosiahnutie vyššej úrovne spoľahlivosti.
 - Zvýšené **testovanie a overovanie systému** je potrebné na presvedčenie klienta systému a regulátorov, že boli dosiahnuté požadované úrovne spoľahlivosti .





Spoľahlivé procesy

- ✧ Na zabezpečenie minimálneho počtu softvérových chýb je dôležité mať dobre definovaný a opakovateľný softvérový proces.
- ✧ Dobre definovaný a opakovateľný proces je taký, ktorý nezávisí úplne od individuálnych zručností; skôr môžu byť uzákonené rôznymi ľuďmi.
- ✧ Regulačné orgány používajú informácie o procese na kontrole, či bola použitá správna prax softvérového inžinierstva.
- ✧ Pre detekciu chýb je jasné, že aktivity procesu by mali zahŕňať značné úsilie venované overovaniu a validácii.

Spoľahlivé charakteristiky procesu



✧ Explicitne definované

- Proces, ktorý má definovaný procesný model, ktorý sa používa na riadenie procesu výroby softvéru. Počas procesu sa musia zbierať údaje, ktoré dokazujú, že vývojový tím dodržal proces, ako je definovaný v modeli procesu.

✧ Opakovateľné

- Proces, ktorý sa nespolieha na individuálny výklad a úsudok. Proces sa môže opakovať v rámci projektov a s rôznymi členmi tímu, bez ohľadu na to, kto sa podielá na vývoji.

Vlastnosti spoľahlivých procesov



Charakteristika procesu	Popis
Auditovateľný	Proces by mal byť zrozumiteľný pre ľudí okrem účastníkov procesu, ktorí môžu kontrolovať dodržiavanie procesných štandardov a podávať návrhy na zlepšenie procesov.
Rozmanitý	Proces by mal zahŕňať nadbytočné a rôznorodé overovacie a validačné činnosti.
Dokumentovateľný	Proces by mal mať definovaný procesný model, ktorý stanovuje činnosti v procese a dokumentáciu, ktorá sa má pri týchto činnostach vytvárať.
Robustný	Proces by mal byť schopný zotaviť sa z porúch jednotlivých činností procesu.
Štandardizovaný	K dispozícii by mal byť komplexný súbor štandardov vývoja softvéru pokryvajúcich výrobu softvéru a dokumentáciu.

Činnosti pre dosiahnutie spoľahlivosti



- ✧ Recenzie požiadaviek: skontrolovať, či sú požiadavky, pokial' je to možné, úplné a konzistentné.
- ✧ Správa požiadaviek: aby sa zabezpečilo, že zmeny požiadaviek sú kontrolované a že je pochopený vplyv navrhovaných zmien požiadaviek.
- ✧ Formálna špecifikácia: kde sa vytvorí a analyzuje matematický model softvéru.
- ✧ Systémové modelovanie: kde je návrh softvéru explicitne zdokumentovaný ako súbor grafických modelov a sú zdokumentované prepojenia medzi požiadavkami a týmito modelmi .

Činnosti pre dosiahnutie spoľahlivosti (2)



- ✧ Kontroly dizajnu a programu: kde sa rôzne popisy systému kontrolujú a kontrolujú ich rôzni ľudia.
- ✧ Statická analýza: kde sa vykonávajú automatizované kontroly zdrojového kódu programu.
- ✧ Plánovanie a správa testov: kde je navrhnutý komplexný súbor testov systému.
 - testovania musí byť starostlivo riadené, aby sa preukázalo, že tieto testy pokrývajú systémové požiadavky a boli správne aplikované v procese testovania.



Kapitola 2 – Softvérrové procesy

Softvérový proces



- ✧ Štruktúrovaný súbor činností potrebných na vývoj softvérového systému.
- ✧ Existuje mnoho rôznych softvérových procesov, ale všetky zahŕňajú:
 - **Špecifikácia** – definovanie toho, čo má systém robiť;
 - **Návrh a implementácia** – definovanie organizácie systému a implementácia systému;
 - **Validácia** – kontrola, či robí to, čo zákazník chce;
 - **Evolúcia** – zmena systému v reakcii na meniace sa potreby zákazníkov.
- ✧ Softvérový procesný model je abstraktná reprezentácia procesu. Predstavuje popis procesu z určitej konkrétnej perspektívy.

Popisy softvérových procesov



- ✧ Keď popisujeme a diskutujeme procesy, zvyčajne hovoríme o činnostiach v týchto procesoch, ako je špecifikácia dátového modelu, návrh používateľského rozhrania atď. a poradie týchto činností.
- ✧ Opis procesov môže tiež zahŕňať:
 - Produkty, ktoré sú výsledkom procesnej činnosti;
 - Roly, ktoré odrážajú zodpovednosť ľudí zapojených do procesu;
 - Predbežné a následné podmienky- čo sú tvrdenia, ktoré sú pravdivé pred a po uzákonení procesnej činnosti alebo po výrobení produktu.

Plánom riadené a agilné procesy



- ✧ Procesy riadené plánom ("plánované procesy") sú procesy, pri ktorých sú všetky procesné činnosti vopred naplánované a pokrok sa meria podľa tohto plánu.
- ✧ V agilných procesoch je plánovanie prírastkové a je jednoduchšie zmeniť proces tak, aby odrážal meniace sa požiadavky zákazníkov.
- ✧ V praxi väčšina procesov zahŕňa prvky plánom riadeného aj agilného prístupu.
- ✧ Neexistujú žiadne správne alebo nesprávne softvérové procesy.



Modely softvérrových procesov

Softvérové procesné modely



✧ Vodopádový model

- Model riadený plánom. Oddelené a odlišné fázy špecifikácie a vývoja.

✧ Postupný vývoj

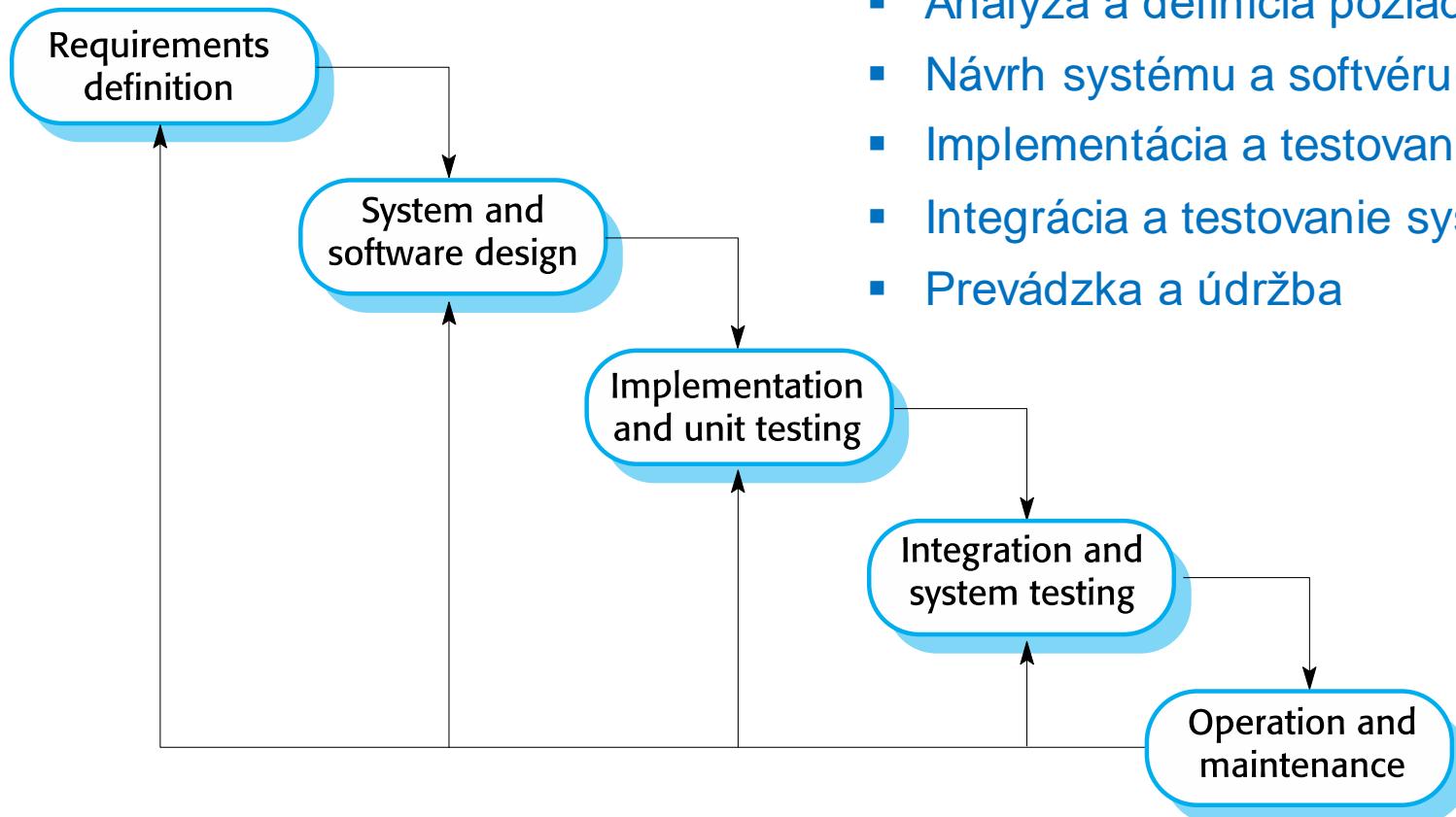
- Špecifikácia, vývoj a validácia sú navzájom prepojené. Môže byť riadený plánom alebo agilný.

✧ Integrácia a konfigurácia

- Systém je zostavený z existujúcich konfigurovateľných komponentov. Môže byť riadený plánom alebo agilný.

✧ V praxi sa väčšina veľkých systémov vyvíja pomocou procesu, ktorý zahŕňa prvky zo všetkých týchto modelov.

(1) Vodopádový model



Fázy

- Analýza a definícia požiadaviek
- Návrh systému a softvéru
- Implementácia a testovanie jednotiek
- Integrácia a testovanie systému
- Prevádzka a údržba

Fázy modelu vodopádu



- ✧ Vo vodopádovom modeli sú oddelené identifikované fázy:
 - Analýza a definícia požiadaviek
 - Návrh systému a softvéru
 - Implementácia a testovanie jednotiek
 - Integrácia a testovanie systému
 - Prevádzka a údržba
- ✧ Hlavnou nevýhodou vodopádového modelu je obtiažnosť prispôsobiť sa zmenám po tom, čo proces prebieha. V zásade musí byť fáza dokončená pred prechodom na ďalšiu fázu.

Problémy s modelom vodopádu



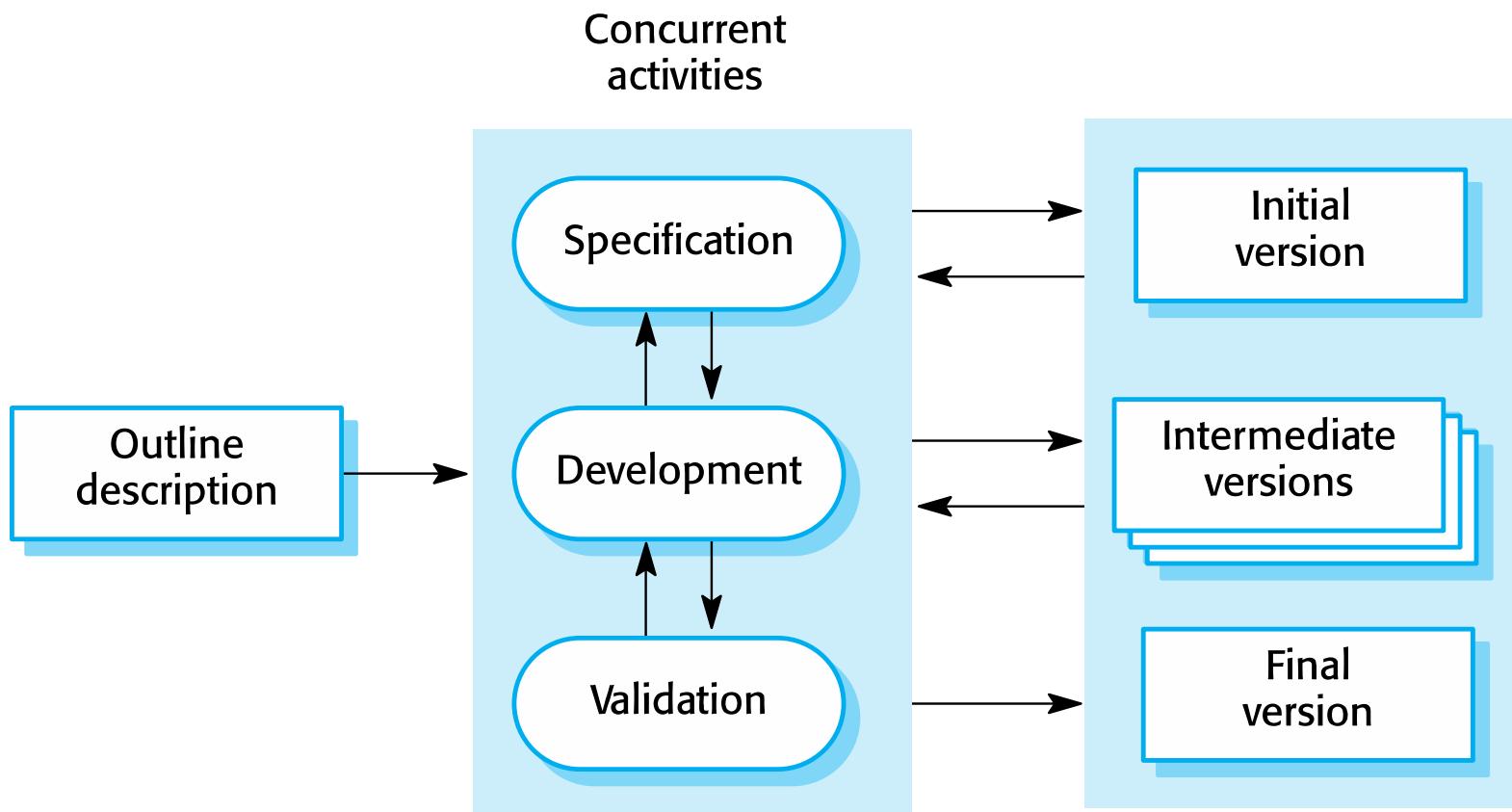
- ✧ Neflexibilné rozdelenie projektu na jednotlivé etapy stŕažuje reakciu na meniace sa požiadavky zákazníkov.
 - Preto je tento model vhodný len vtedy, keď sú požiadavky dobre pochopené a zmeny budú počas procesu návrhu značne obmedzené.
 - Len málo obchodných systémov má stabilné požiadavky.
- ✧ Vodopádový model sa väčšinou používa pre veľké projekty systémového inžinierstva, kde sa systém vyvíja na niekoľkých miestach.
 - Za týchto okolností pomáha plánom riadený charakter modelu vodopádu koordinovať prácu.

(2) Postupný vývoj



Software Engineering
6th edition

Ian Sommerville



Výhody postupného vývoja



- ✧ Znižujú sa náklady na prispôsobenie sa meniacim sa požiadavkám zákazníkov.
 - Množstvo analýz a dokumentácie, ktoré je potrebné prepracovať, je oveľa menšie, ako sa vyžaduje pri modeli vodopádu.
- ✧ Je jednoduchšie získať spätnú väzbu od zákazníkov na vykonanú vývojovú prácu.
 - Zákazníci môžu komentovať ukážky softvéru a vidieť, koľko sa implementovalo.
- ✧ Je možné rýchlejšie dodanie a nasadenie užitočného softvéru zákazníkovi.
 - Zákazníci môžu softvér používať a získavať z neho hodnotu skôr, ako je to možné pri vodopádovom procese.

Problémy s postupným vývojom



- ✧ Proces nie je viditeľný.
 - Manažéri potrebujú pravidelné výstupy na meranie pokroku. Ak sa systémy vyvíjajú rýchlo, nie je nákladovo efektívne vytvárať dokumenty, ktoré odrážajú každú verziu systému.
- ✧ Štruktúra systému má tendenciu degradovať, keď sa pridávajú nové prírastky.
 - Pokiaľ nie je vynaložený čas a peniaze na refaktorovanie na zlepšenie softvéru, pravidelné zmeny majú tendenciu poškodiť jeho štruktúru. Začlenenie ďalších softvérových zmien je čoraz ťažšie a nákladnejšie.

(3) Integrácia a konfigurácia



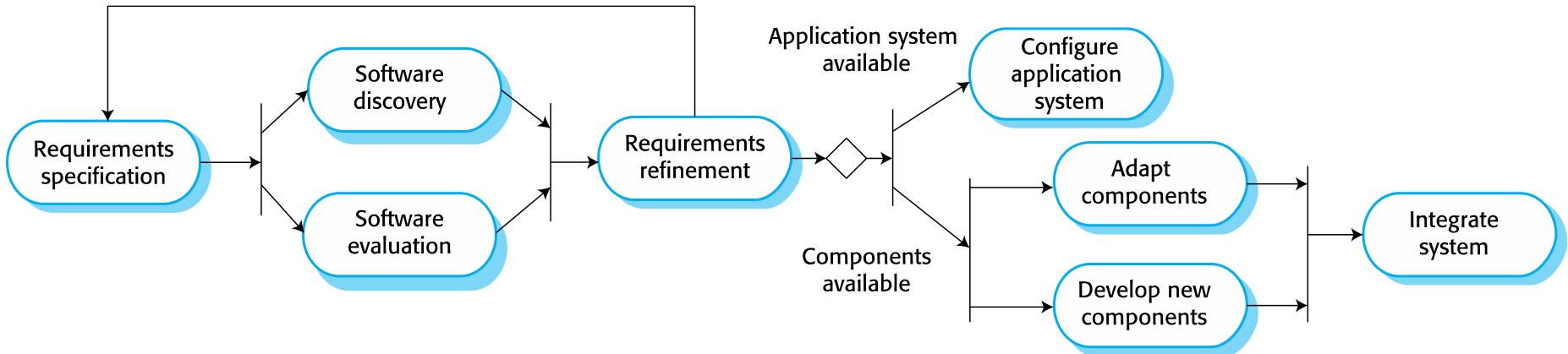
- ✧ Založené na opäťovnom použití softvéru, kde sú systémy integrované z existujúcich komponentov alebo aplikačných systémov (niekedy nazývaných COTS – Commercial-off-the-shelf) systémov).
- ✧ Opäťovne použité prvky môžu byť nakonfigurované tak, aby sa ich správanie a funkčnosť prispôsobili požiadavkám používateľa
- ✧ Opäťovné použitie je v súčasnosti štandardným prístupom k budovaniu mnohých typov podnikových systémov
 - Opäťovné použitie podrobnejšie popísané v kapitole 15, prednáška 9.

Typy opäťovne použiteľného softvéru



- ✧ Samostatné aplikačné systémy (niekedy nazývané COTS), ktoré sú nakonfigurované na použitie v konkrétnom prostredí.
- ✧ Kolekcie objektov, ktoré sú vyvinuté ako balík na integráciu s komponentovým rámcem, ako je .NET alebo J2EE.
- ✧ Webové služby, ktoré sú vyvinuté podľa servisných štandardov a ktoré sú dostupné pre vzdialené vyvolanie.

Softvérové inžinierstvo orientované na opäťovné použitie



Kľúčové fázy procesu

- ✧ Špecifikácia požiadaviek
- ✧ Objav a vyhodnotenie nájdeného softvéru
- ✧ Spresnenie požiadaviek
- ✧ Konfigurácia aplikačného systému
- ✧ Prispôsobenie a integrácia komponentov



Výhody a nevýhody

- ✧ Znížené náklady a riziká, pretože menej softvéru sa vyvíja od nuly
- ✧ Rýchlejšie dodanie a nasadenie systému
- ✧ Kompromisy požiadaviek sú však nevyhnutné, takže systém nemusí splňať skutočné potreby používateľov
- ✧ Strata kontroly nad vývojom opäťovne použitých prvkov systému



Procesné činnosti

Procesné činnosti



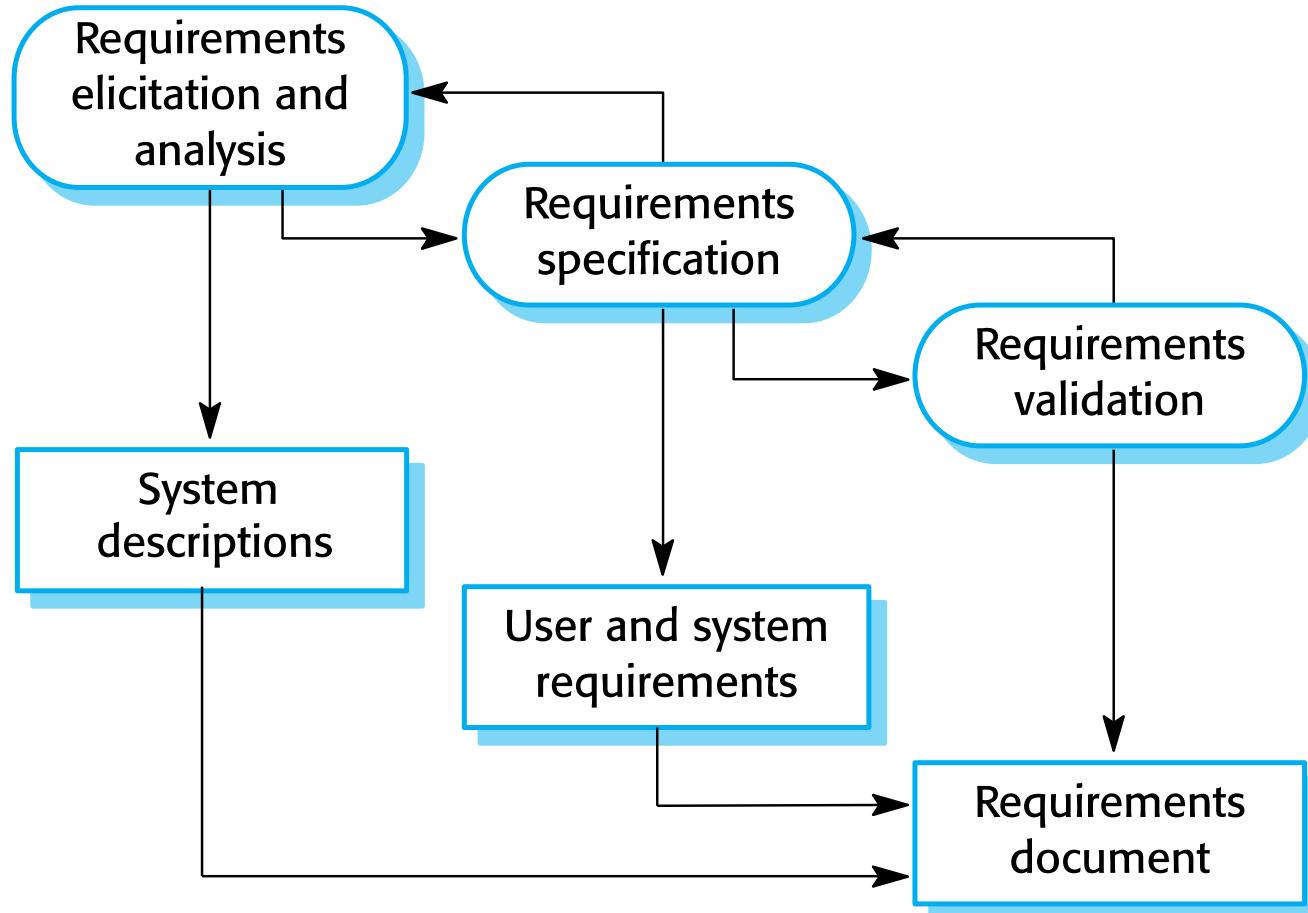
- ✧ Reálne softvérové procesy sú vzájomne prepojené sekvencie technických, kolaboratívnych a manažérskych činností s celkovým cieľom špecifikovať, navrhnúť, implementovať a otestovať softvérový systém.
- ✧ Štyri základné procesné činnosti ("aktivity softvérového procesu") sú **špecifikácie, vývoja, validácie a evolúcie** a sú v rôznych vývojových procesoch organizované odlišne.
- ✧ Napríklad vo vodopádovom modeli sú usporiadané v poradí, zatiaľ čo v prírastkovom vývoji sú vykonávané súčasne.

(1) Špecifikácia softvéru



- ✧ Proces zistovania, aké služby sú požadované a obmedzenia fungovania a rozvoja systému.
- ✧ Požiadavky na inžiniersky proces (viď obr.)
 - Získavanie a analýza požiadaviek
 - Čo od systému požadujú alebo očakávajú účastníci systému?
 - Špecifikácia požiadaviek
 - Detailné definovanie požiadaviek
 - Overenie požiadaviek
 - Kontrola platnosti požiadaviek

Analýza s špecifikácia požiadaviek



(2) Návrh a implementácia softvéru



✧ Proces prevodu špecifikácie systému na spustiteľný systém.

✧ Návrh softvéru

- Navrhnite softvérovú štruktúru, ktorá realizuje špecifikáciu;

✧ Implementácia

- Preložte túto štruktúru do spustiteľného programu;

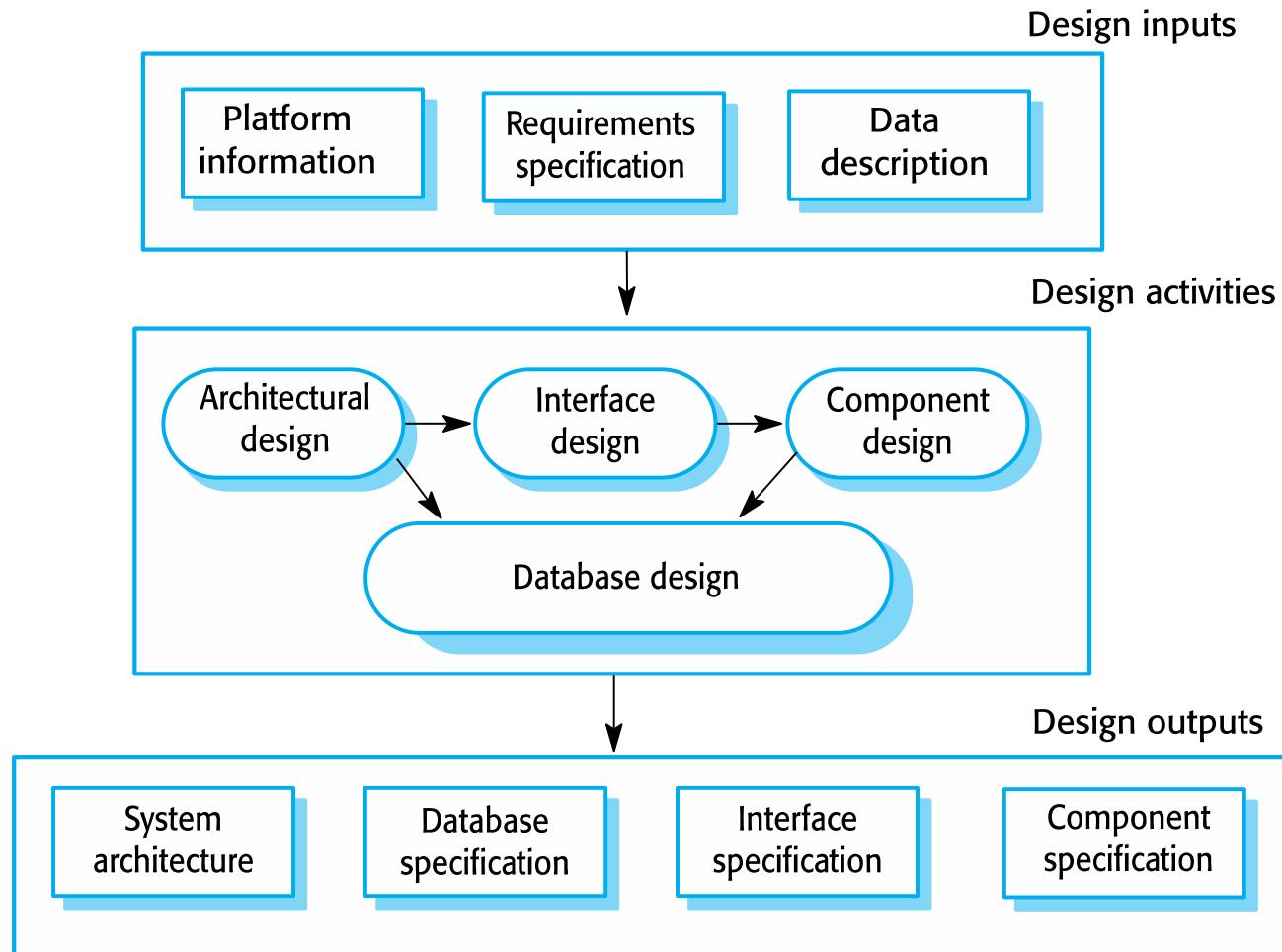
✧ Činnosti návrhu a implementácie spolu úzko súvisia a môžu sa navzájom dopĺňať.

(2a) Dizajnérské činnosti



- ✧ *Architektonický návrh*, kde identifikujete celkovú štruktúru systému, hlavné komponenty (subsystémy alebo moduly), ich vzťahy a spôsob ich rozloženia.
- ✧ *Návrh databázy*, kde navrhujete štruktúry údajov systému a ako majú byť reprezentované v databáze.
- ✧ *Návrh rozhrania*, kde definujete rozhrania medzi komponentmi systému.
- ✧ *Výber a dizajn komponentov*, kde opakovane hľadáte použiteľné komponenty. Ak nie sú k dispozícii, navrhnite, ako systém bude fungovať.

Všeobecný model procesu navrhovania



(2b) Implementácia systému



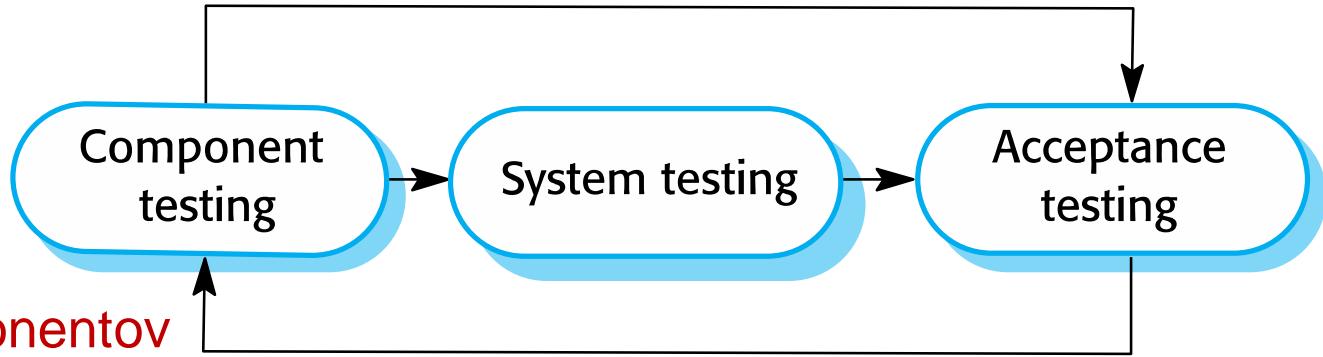
- ✧ Softvér sa implementuje buď vývojom programu alebo programov alebo konfiguráciou aplikačného systému.
- ✧ Návrh a implementácia sú prepojené činnosti pre väčšinu typov softvérových systémov.
- ✧ Programovanie je individuálna činnosť bez štandardného procesu.
- ✧ Ladenie je činnosť zameraná na vyhľadávanie chýb programu a nápravu týchto chýb.



(3) Validácia softvéru

- ✧ **Verifikácia a validácia (V & V)** je určená na preukázanie, že systém zodpovedá svojej špecifikácii (verifikácia) a splňa požiadavky zákazníka systému (validácia).
- ✧ Zahŕňa procesy kontroly, preskúmania a testovanie systému.
- ✧ **Testovanie systému** zahŕňa spustenie systému pomocou testovacích prípadov, ktoré sú odvodené zo špecifikácie skutočných údajov, ktoré má systém spracovať.
- ✧ **Testovanie** je najčastejšie používaná V & V aktivita.

Fázy testovania



✧ Testovanie komponentov

- Jednotlivé komponenty sú testované nezávisle;
- Komponenty môžu byť funkcie alebo objekty alebo koherentné zoskupenia týchto entít.

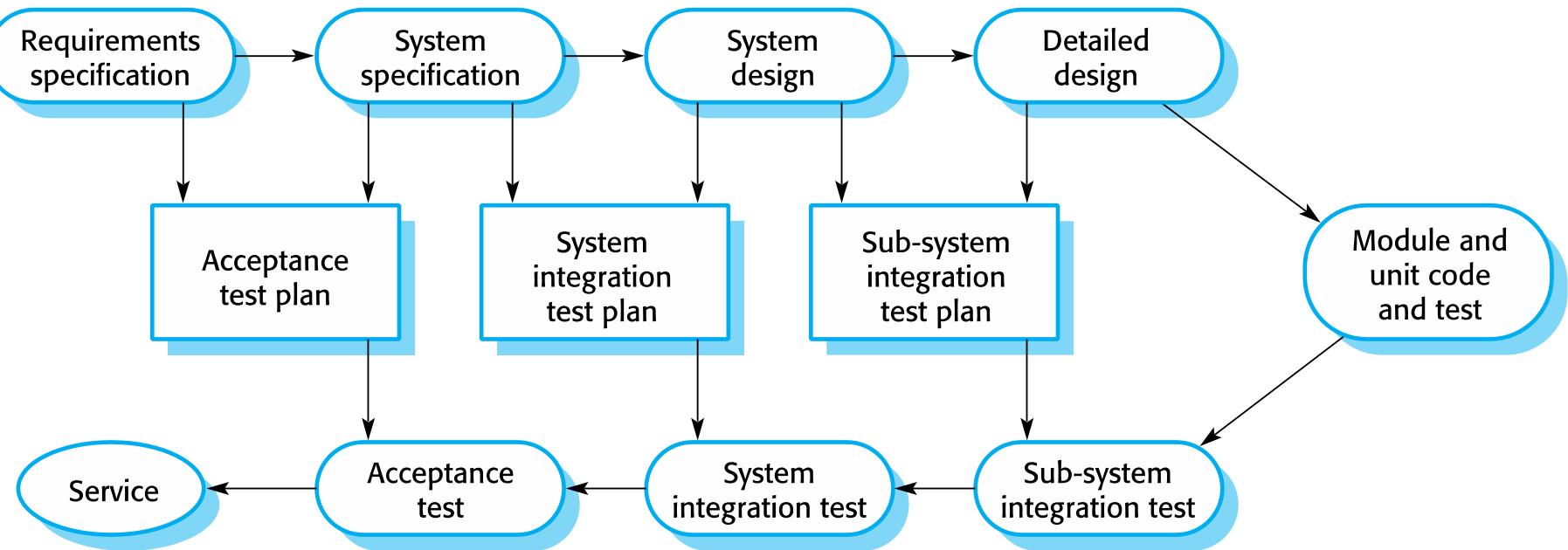
✧ Testovanie systému

- Testovanie systému ako celku. Zvlášť dôležité je testovanie emergentných vlastností.

✧ Zákaznícke testovanie

- Testovanie s údajmi od zákazníkov na kontrolu, či systém spĺňa potreby zákazníka.

Fázy testovania v softvérovom procese riadenom plánom (V-model)

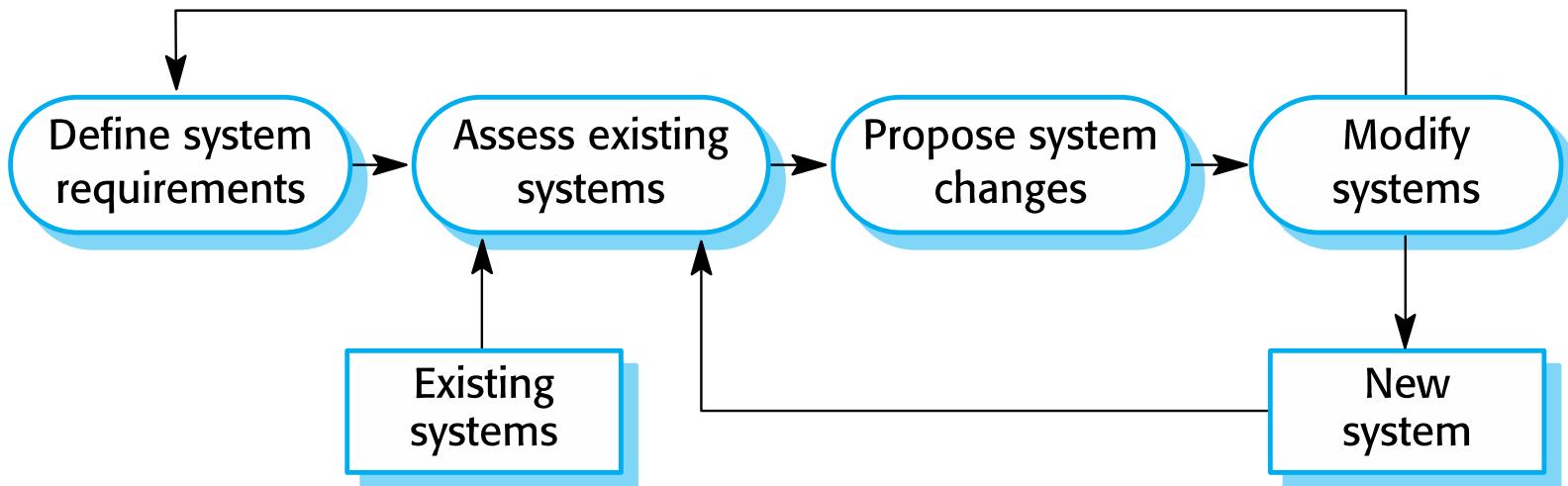




(4) Evolúcia alebo ďalší vývoj softvéru

- ✧ Softvér je vo svojej podstate flexibilný a môže sa meniť.
- ✧ Ked'že požiadavky sa menia v dôsledku meniacich sa obchodných podmienok, musí sa vyvíjať a meniť aj softvér, ktorý podporuje podnikanie.
- ✧ Hoci existuje hranica medzi vývojom a evolúciou (údržbou), je to stále viac irelevantné, ked'že stále menej systémov je úplne nových.

Evolúcia systému





Vyrovnanie sa so zmenou



Vyrovnanie sa so zmenou

- ✧ Zmena je nevyhnutná vo všetkých veľkých softvérových projektoch.
 - Obchodné zmeny vedú k novým a zmeneným systémovým požiadavkám
 - Nové technológie otvárajú nové možnosti na zlepšenie implementácií
 - Meniace sa platformy si vyžadujú zmeny aplikácií
- ✧ Zmena viedie k prepracovaniu, takže náklady na zmenu zahŕňajú prepracovanie (napr. prehodnotenie požiadaviek), ako aj náklady na implementáciu novej funkcionality.

Zníženie nákladov na prepracovanie



- ✧ **Predvídanie zmien**, kde softvérový proces zahŕňa činnosti, ktoré môžu predvídať možné zmeny predtým, ako je potrebné vykonať významné prepracovanie.
 - Napríklad môže byť vyvinutý prototyp systému, ktorý zákazníkom ukáže niektoré kľúčové vlastnosti systému.
- ✧ **Tolerancia zmien**, kde je proces navrhnutý tak, aby sa zmeny mohli prispôsobiť relatívne nízkym nákladom.
 - To zvyčajne zahŕňa určitú formu postupného vývoja. Navrhované zmeny môžu byť implementované v prírastkoch, ktoré ešte neboli vyvinuté. Ak to nie je možné, môže byť zmenený iba jeden prírastok (malá časť systému), aby sa začlenila zmena.

Vyrovnanie sa s meniacimi sa požiadavkami



- ✧ **Systémové prototypovanie** , kde sa rýchlo vyvinie verzia systému alebo časť systému, aby sa preverili požiadavky zákazníka a uskutočniteľnosť návrhových rozhodnutí. Tento prístup podporuje predvídanie zmien.
- ✧ **Inkrementálne doručovanie** , kde sa systémové inkrementy doručujú zákazníkovi na pripomienkovanie a experimentovanie. To podporuje vyhýbanie sa zmenám a toleranciu zmien.

Prototypovanie softvéru



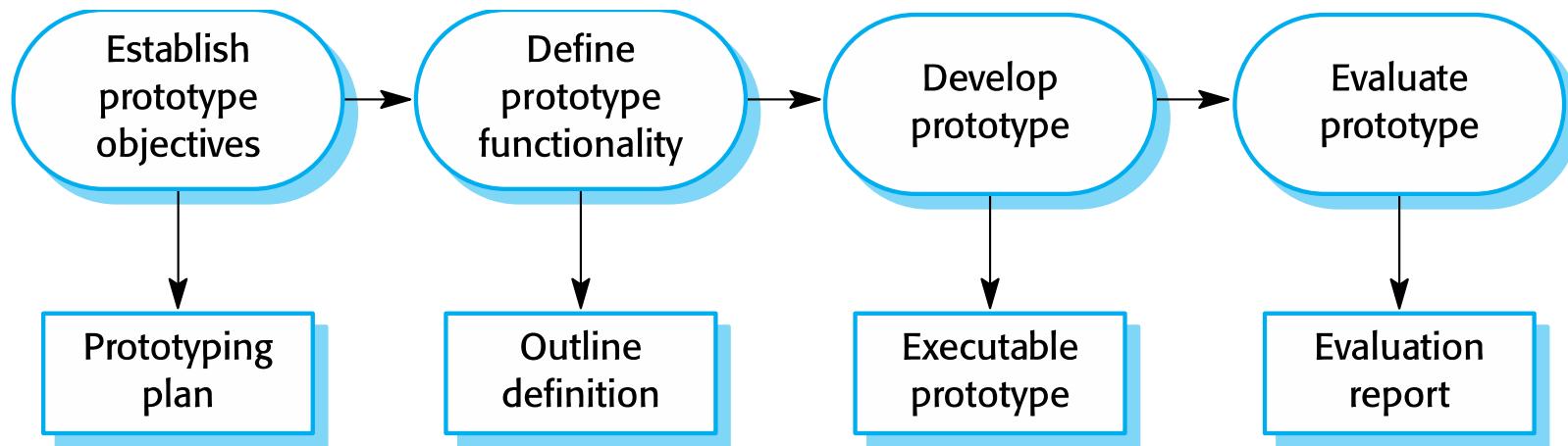
- ✧ Prototyp je počiatočná verzia systému používaná na demonštráciu konceptov a vyskúšanie možností dizajnu.
- ✧ Prototyp je možné použiť:
 - Proces analýzy požiadaviek na pomoc pri vytváraní a overovaní požiadaviek;
 - V procese návrhu preskúmať možnosti a vyvinúť dizajn používateľského rozhrania;
 - V procese testovania na spustenie testov

Výhody prototypovania



- ✧ Vylepšená použiteľnosť systému.
- ✧ Presnejšie prispôsobenie skutočným potrebám používateľov.
- ✧ Vylepšená kvalita dizajnu.
- ✧ Vylepšená udržiavateľnosť.
- ✧ Znížené úsilie o vývoj.

Proces vývoja prototypu



Vývoj prototypu



- ✧ Môžu byť založené na jazykoch alebo nástrojoch rýchleho prototypovania
- ✧ Môže zahŕňať vynechanie funkcií ("nevýhody")
 - Prototyp by sa mal zameráť na oblasti produktu, ktoré nie sú dobre pochopené;
 - Prototyp nemusí obsahovať kontrolu a obnovu chýb;
 - Zamerajte sa skôr na funkčné ako nefunkčné požiadavky, ako je spoľahlivosť a bezpečnosť

Vyhadzovacie prototypy



- ✧ Prototypy by sa mali po vývoji vyradiť, pretože nie sú dobrým základom pre produkčný systém:
 - Môže byť nemožné vyladiť systém tak, aby spĺňal nefunkčné požiadavky;
 - Prototypy sú zvyčajne nezdokumentované;
 - Štruktúra prototypu je zvyčajne degradovaná rýchloou zmenou;
 - Prototyp pravdepodobne nebude spĺňať bežné organizačné štandardy kvality.

"nevýhody"

Prírastkové doručenie



- ✧ Namiesto dodania systému ako jednej dodávky je vývoj a dodávka rozčlenená na prírastky, pričom každý prírastok poskytuje časť požadovanej funkčnosti.
- ✧ Požiadavky používateľov sú uprednostňované a požiadavky s najvyššou prioritou sú zahrnuté v prvých prírastkoch.
- ✧ Po spustení vývoja prírastku sa požiadavky zmrazia, hoci požiadavky na neskoršie prírastky sa môžu nadalej vyvíjať.

Postupný vývoj a doručovanie



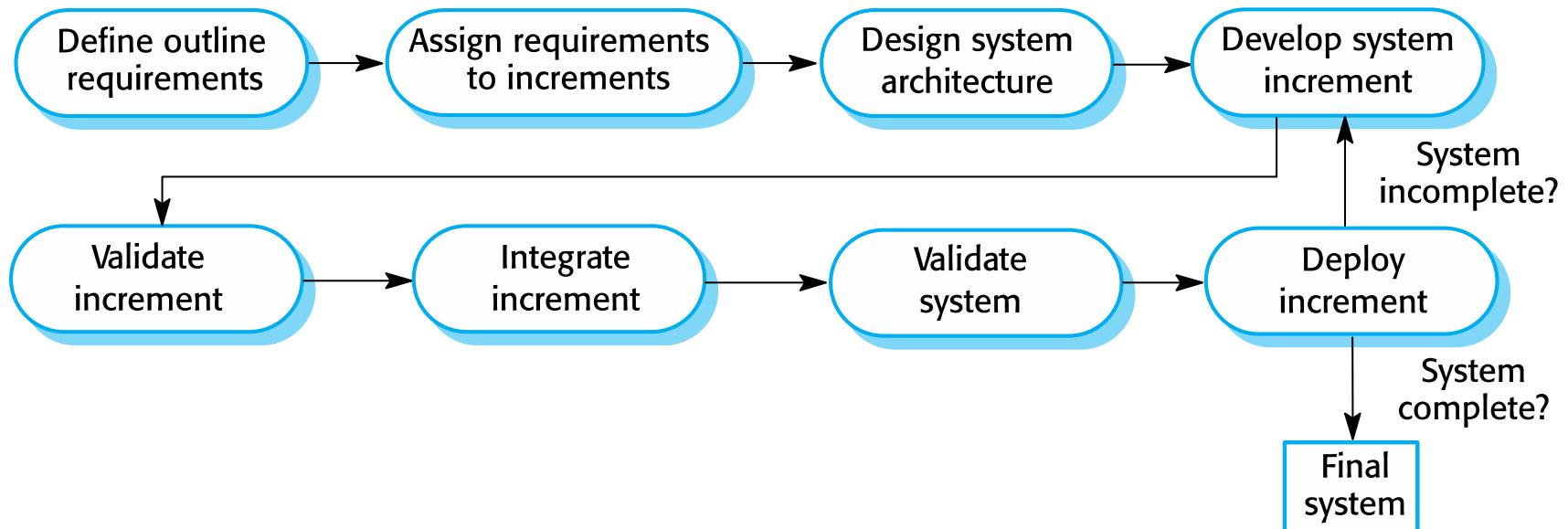
✧ Postupný vývoj

- Vyvíjajte systém v prírastkoch a vyhodnoťte každý prírastok predtým, ako pristúpite k vývoju ďalšieho prírastku;
- Normálny prístup používaný v agilných metódach;
- Hodnotenie vykonáva používateľ/zástupca zákazníka.

✧ Prírastkové doručenie

- Nasadťte prírastok, aby bol používaný koncovými používateľmi;
- Realistickejšie hodnotenie praktického používania softvéru;
- Je ťažké toto realizovať pri nahradzovaní systému, pretože prírastky majú menšiu funkčnosť ako vymieňaný systém.

Prírastkové doručenie



Výhody prírastkového doručovania



- ✧ Hodnota pre zákazníka môže byť poskytovaná s každým prírastkom, takže funkčnosť systému je k dispozícii skôr.
- ✧ Skoré prírastky fungujú ako prototyp, ktorý pomáha získať požiadavky na neskoršie prírastky.
- ✧ Nižšie riziko celkového zlyhania projektu.
- ✧ Systémové služby s najvyššou prioritou sú zvyčajne najviac testované.

Problémy s postupným doručovaním



- ✧ Väčšina systémov vyžaduje "základy", ktoré používajú rôzne časti systému.
 - Ked'že požiadavky nie sú detailne definované, kým sa nezačne implementovať prírastok, môže byť ľažké identifikovať spoločné služby, ktoré sú potrebné pre všetky prírastky.
- ✧ Podstatou iteračných procesov je, že špecifikácia sa vytvára spolu so softvérom.
 - To je však v rozpore s modelom obstarávania mnohých organizácií, kde je kompletná špecifikácia systému súčasťou zmluvy o vývoji systému.



Vylepšenie procesov

Vylepšenie procesu



- ✧ Mnoho softvérových spoločností sa obrátilo na zlepšovanie softvérových procesov ako na spôsob zvyšovania kvality svojho softvéru, znižovania nákladov alebo urýchlenia procesov vývoja.
- ✧ Zlepšenie procesov znamená pochopenie existujúcich procesov a zmenu týchto procesov s cieľom zvýšiť kvalitu produktu a/alebo znížiť náklady a čas vývoja.

Prístupy k zlepšeniu



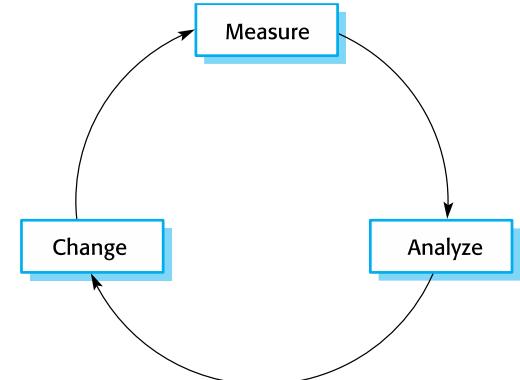
- ✧ Prístup procesnej zrelosti, ktorý sa zameriava na zlepšovanie procesného a projektového riadenia a zavádzanie dobrej praxe softvérového inžinierstva.
 - Úroveň zrelosti procesov odráža mieru, do akej bola v procesoch vývoja softvéru organizácie prijatá dobrá technická a manažérská prax.
- ✧ Agilný prístup, ktorý sa zameriava na iteratívny vývoj a znížovanie režijných nákladov v softvérovom procese.
 - Primárhou charakteristikou agilných metód je rýchle poskytovanie funkčnosti a schopnosť reagovať na meniace sa požiadavky zákazníkov.

Činnosti na zlepšenie procesov



✧ Procesné meranie

- Meriate jeden alebo viac atribútov softvérového procesu alebo produktu. Tieto merania tvoria základ, ktorý vám pomôže rozhodnúť, či boli zlepšenia procesov efektívne.



✧ Procesná analýza

- Súčasný proces sa hodnotí a identifikujú sa slabé miesta a úzke miesta procesu. Môžu byť vyvinuté modely procesov (niekedy nazývané mapy procesov), ktoré popisujú proces .

✧ Zmena

- procesné zmeny na odstránenie niektorých zistených slabých stránok procesov. Tieto sa zavedú a cyklus sa obnoví na zber údajov o účinnosti zmien .

Procesné meranie



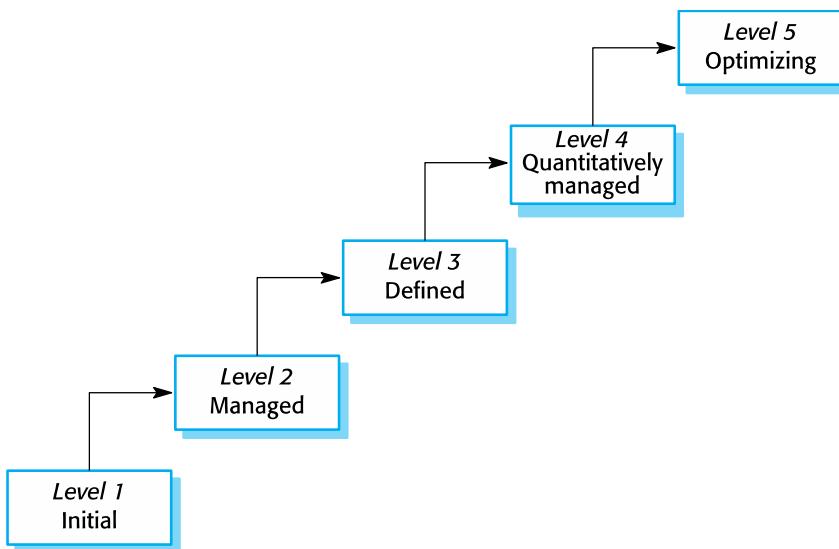
- ✧ Mali by sa zbierať kvantitatívne údaje o procese
 - Ak však organizácie nemajú jasne definované procesné štandardy, je to veľmi ťažké, pretože neviete, čo merať. Pred možným meraním môže byť potrebné definovať proces.
- ✧ Procesné merania by sa mali používať na posúdenie zlepšenia procesov
 - To však neznamená, že merania by mali viest' k zlepšeniam. Motorom zlepšovania by mali byť organizačné ciele.

Procesné metriky



- ✧ Čas potrebný na dokončenie procesných činností
 - Napr. čas alebo úsilie v kalendári na dokončenie činnosti alebo procesu.
- ✧ Zdroje potrebné na procesy alebo činnosti
 - Napr. celkové úsilie v osobo-dňoch.
- ✧ Počet výskytov konkrétnej udalosti
 - Napr. počet zistených defektov.

Procesy - úrovne zrelosti spôsobilosti



5. Optimalizované - Stratégie zlepšovania procesov definované a používané
4. Organizované - Definované a používané stratégie manažérstva kvality
3. Definované a používané postupy a stratégie procesného riadenia
2. Opakovateľné - Definované a používané postupy riadenia produktu
1. Počiatočné - V podstate nekontrolované



Kapitola 22 – Projektový manažment

Kapitola 23 – Plánovanie projektu

Riadenie softvérových projektov



- ✧ Zaoberá sa činnosťami súvisiacimi so zabezpečením dodania softvéru včas a podľa plánu a v súlade s požiadavkami organizácií vyvýjajúcich a obstarávajúcich softvér.
- ✧ Projektový manažment je potrebný, pretože vývoj softvéru vždy podlieha rozpočtovým a časovým obmedzeniam, ktoré stanovuje organizácia, ktorá softvér vyvíja.



Kritériá úspešnosti

- ✧ Doručte softvér zákazníkovi v dohodnutom čase.
- ✧ Udržujte celkové náklady v rámci rozpočtu.
- ✧ Dodávajte softvér, ktorý spĺňa očakávania zákazníka.
- ✧ Udržujte koherentný a dobre fungujúci vývojový tím.



Rozdiely v správe softvéru

✧ Produkt je **nehmotný**

- Softvér nie je vidieť ani sa ho nemožno dotknúť. Manažéri softvérových projektov nemôžu vidieť pokrok jednoduchým pohľadom na artefakt, ktorý sa vytvára.

✧ Mnohé softvérové projekty sú „**jednorazové**“ projekty

- Veľké softvérové projekty sa zvyčajne v niektorých smeroch líšia od predchádzajúcich projektov. Dokonca aj manažéri, ktorí majú veľa predchádzajúcich skúseností, môžu mať problém predvídať problémy.

✧ Softvérové procesy sú variabilné a organizačne špecifické

- Stále nemôžeme spoľahlivo predpovedať, kedy konkrétny softvérový proces pravdepodobne povedie k problémom s vývojom.

Faktory ovplyvňujúce riadenie projektu



- ✧ Veľkosť spoločnosti
- ✧ Zákazníci softvéru
- ✧ Veľkosť softvéru
- ✧ Typ softvéru
- ✧ Organizačná kultúra
- ✧ Procesy vývoja softvéru

Tieto faktory znamenajú, že projektoví manažéri v rôznych organizáciách môžu pracovať celkom odlišným spôsobom.

Univerzálne manažérske činnosti



✧ *Plánovanie projektu*

- Za plánovanie sú zodpovední projektoví manažéri. Odhadovanie a plánovanie vývoja projektu a pridelovanie ľudí k úlohám.

✧ *Riadenie rizík*

- Projektoví manažéri posudzujú riziká, ktoré môžu ovplyvniť projekt, monitorujú tieto riziká a prijímajú opatrenia, keď nastanú problémy.

✧ *Riadenie ľudí*

- Projektoví manažéri si musia vybrať ľudí do svojho tímu a zaviesť spôsoby práce, ktoré vedú k efektívному tímovému výkonu.

(Ďalšie) Manažérske činnosti



✧ Nahlasovanie

- Projektoví manažéri sú zvyčajne zodpovední za podávanie správ o priebehu projektu zákazníkom a manažérom spoločnosti vyvíjajúcej softvér.

✧ Písanie návrhu

- Prvá fáza softvérového projektu môže zahŕňať napísanie návrhu na získanie zákazky na vykonanie diela. Návrh popisuje ciele projektu a spôsob jeho realizácie.



Plánovanie projektu

Plánovanie projektu



- ✧ **Plánovanie projektu** zahŕňa rozdelenie práce na časti a pridelenie ich členom projektového tímu, predvídanie problémov, ktoré môžu nastať, a prípravu predbežných riešení týchto problémov.
- ✧ **Projektový plán**, ktorý sa vytvára na začiatku projektu, sa používa na informovanie projektového tímu a zákazníkov o tom, ako bude práca vykonaná, a na pomoc pri hodnotení pokroku na projekte.

Etapy plánovania



- ✧ Vo fáze návrhu , keď sa uchádzate o zákazku na vývoj alebo poskytovanie softvérového systému.
- ✧ Počas fázy spustenia projektu , keď musíte naplánovať, kto bude na projekte pracovať, ako bude projekt rozdelený na prírastky, ako sa budú pridelovať zdroje v rámci vašej spoločnosti atď.
- ✧ Pravidelne počas celého projektu , keď svoj plán upravíte na základe získaných skúseností a informácií z monitorovania postupu prác.

Plánovanie návrhu



- ✧ Plánovanie môže byť potrebné iba s rámcovými požiadavkami na softvér.
- ✧ Cieľom plánovania v tejto fáze je poskytnúť zákazníkom informácie, ktoré sa použijú pri stanovovaní ceny za systém.
- ✧ Cena projektu zahŕňa odhad, kolko bude stáť vývoj softvéru, pričom sa zohľadnia faktory, ako sú náklady na zamestnancov, náklady na hardvér, náklady na softvér atď.

Plánovanie spustenia projektu



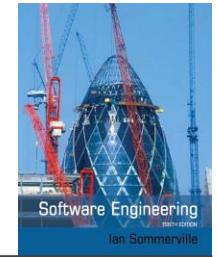
- ✧ V tejto fáze viete viac o systémových požiadavkách, ale nemáte informácie o návrhu alebo implementácii
- ✧ Vytvorte dostatočne podrobný plán na rozhodovanie o rozpočte projektu a personálnom obsadení.
 - Tento plán je základom pre pridelovanie zdrojov projektu
- ✧ Plán spustenia by mal definovať aj mechanizmy monitorovania projektu
- ✧ Pre agilný vývoj je stále potrebný plán spustenia, aby bolo možné prideliť zdroje na projekt

Plánovanie vývoja



- ✧ Plán projektu by sa mal pravidelne upravovať podľa toho, ako projekt postupuje a vy viete viac o softvéri a jeho vývoji
- ✧ Harmonogram projektu, odhad nákladov a riziká sa musia pravidelne revidovať

Plánom riadený vývoj



- ✧ **Plánom riadený alebo plánom založený vývoj** je prístup k softvérovému inžinierstvu, kde je proces vývoja podrobne naplánovaný.
 - Plánom riadený vývoj je založený na technikách riadenia inžinierskych projektov a je „tradičným“ spôsobom riadenia veľkých projektov vývoja softvéru.
- ✧ **plán projektu**, ktorý zaznamenáva prácu, ktorá sa má vykonáť, kto ju bude vykonávať, harmonogram vývoja a pracovné produkty.
- ✧ Manažéri využívajú plán na podporu rozhodovania o projekte a ako spôsob merania pokroku.

Plánom riadený vývoj – klady a zápory



- ✧ Argumenty v prospech prístupu založeného na pláne sú, že včasné plánovanie umožňuje dôsledné zohľadnenie organizačných problémov (dostupnosť zamestnancov, iné projekty atď.) a že potenciálne problémy a závislosti sa odhalia skôr, než sa projekt začne / kým sa projekt rozbehne.
- ✧ Hlavným argumentom proti plánom riadenému vývoju je, že mnohé skoré rozhodnutia musia byť revidované kvôli zmenám v prostredí, v ktorom sa má softvér vyvíjať a používať.

Projektové plány



- ✧ V plánom riadenom rozvojovom projekte **projektový plán** stanovuje **zdroje (kto)** dostupné pre projekt, **rozpis prác (čo)** a **harmonogram (kedy)** vykonávania práce.
- ✧ Plán úsekov
 - Úvod
 - Organizácia projektu
 - Analýza rizík
 - Požiadavky na hardvérové a softvérové zdroje
 - Rozpis práce
 - Harmonogram projektu
 - Mechanizmy monitorovania a podávania správ

Doplnky projektového plánu



Plán	Popis
Plán riadenia konfigurácie	Opisuje postupy a štruktúry riadenia konfigurácie, ktoré sa majú použiť.
Plán nasadenia	Popisuje, ako bude softvér a súvisiaci hardvér (ak je potrebný) nasadený v prostredí zákazníka. To by malo zahŕňať plán migrácie údajov z existujúcich systémov.
Plán údržby	Predpovedá požiadavky na údržbu, náklady a úsilie.
Plán kvality	Popisuje postupy a štandardy kvality, ktoré sa použijú v projekte.
Validačný plán	Opisuje prístup, zdroje a plán používaný na overenie systému.



Proces plánovania

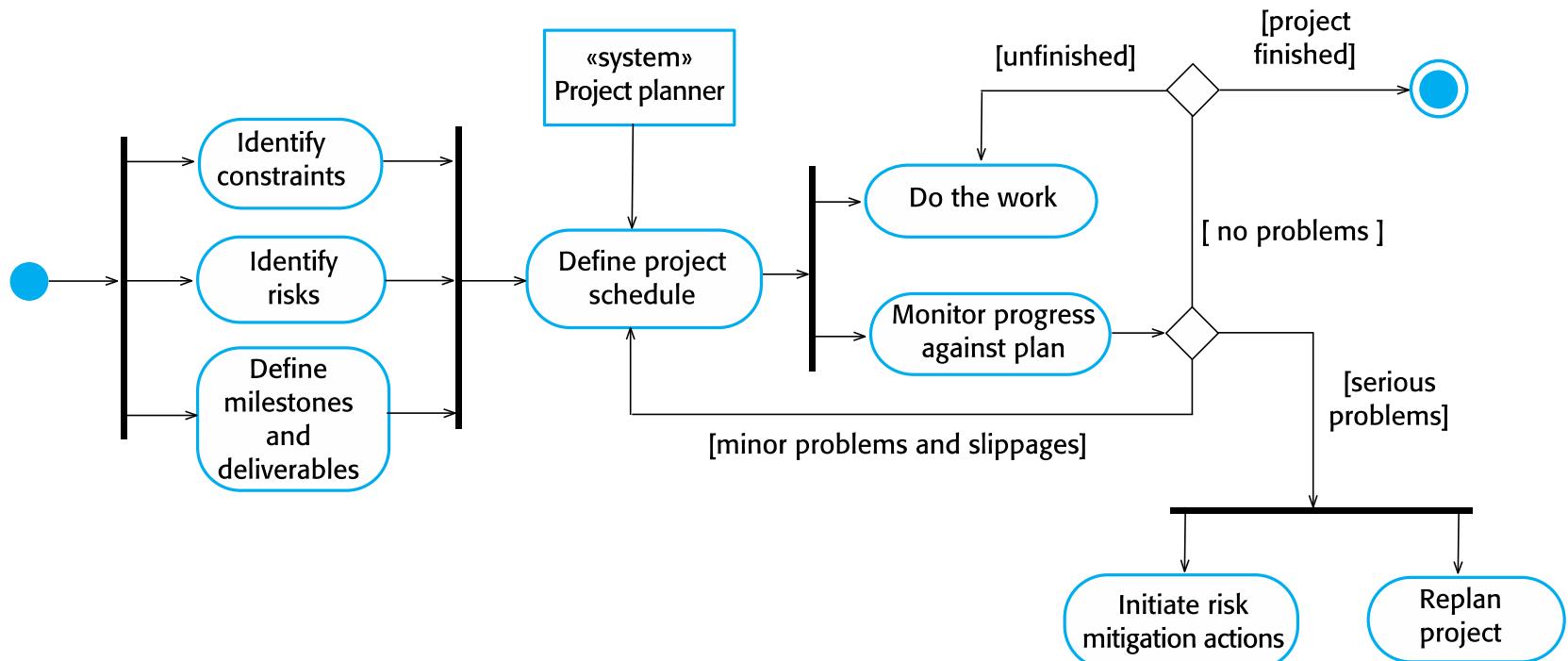
- ✧ Plánovanie projektu je iteratívny proces, ktorý sa začína vytvorením počiatočného plánu projektu počas fázy spustenia projektu.
- ✧ Zmeny plánu sú nevyhnutné.
 - Keď bude počas projektu k dispozícii viac informácií o systéme a projektovom tíme, mali by ste plán pravidelne revidovať, aby odrážal zmeny požiadaviek, harmonogramu a rizík.
 - Zmena obchodných cieľov vedie aj k zmenám v projektových plánoch. Keď sa obchodné ciele zmenia, môže to ovplyvniť všetky projekty, ktoré potom možno bude potrebné preplánovať.

Proces plánovania projektu



Software Engineering
6th edition

Ian Sommerville



Plánovacie predpoklady



- ✧ Pri definovaní plánu projektu by ste mali robiť skôr realistické ako optimistické predpoklady.
- ✧ Počas projektu sa vždy vyskytnú nejaké problémy a vedú k omeškaniu
- ✧ Vaše počiatočné predpoklady a plánovanie by preto mali zohľadňovať neočakávané problémy.
- ✧ Mali by ste zahrnúť nepredvídane udalosti, aby ak sa niečo pokazilo, aby váš harmonogram doručenia nebol vážne narušený.

Plánovanie projektu



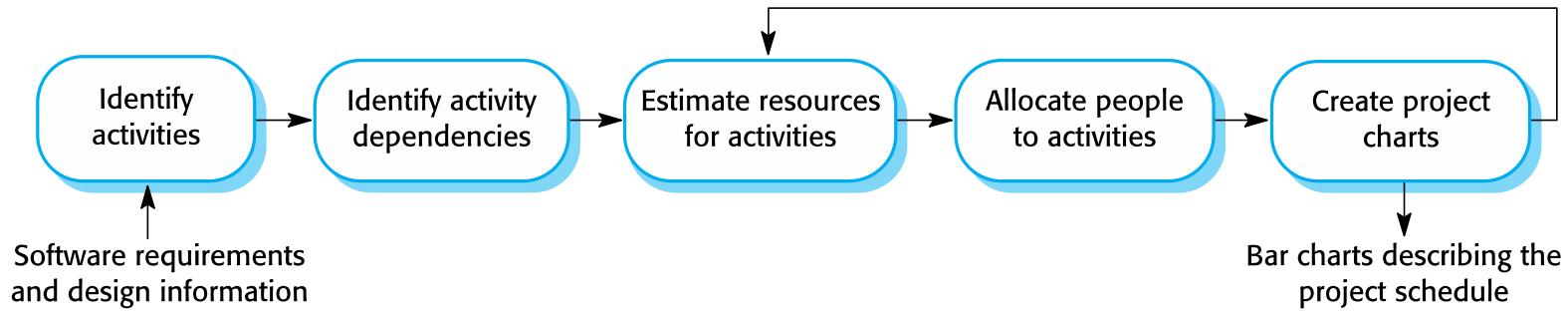
- ✧ **Plánovanie projektu** je proces rozhodovania o tom, ako bude práca v projekte organizovaná ako samostatné úlohy a kedy a ako sa tieto úlohy vykonajú.
- ✧ Odhadujete kalendárny čas potrebný na dokončenie každej úlohy, potrebné úsilie a kto bude pracovať na úlohách, ktoré boli identifikované.
- ✧ Musíte tiež odhadnúť zdroje potrebné na dokončenie každej úlohy, ako je miesto na disku požadované na serveri, čas potrebný na špecializovaný hardvér, ...

Plánovanie



- ✧ Rozdeľte projekt na úlohy a odhadnite čas a zdroje potrebné na dokončenie každej úlohy.
- ✧ Organizujte úlohy súbežne, aby ste optimálne využili pracovnú silu.
- ✧ Minimalizujte závislosti úloh, aby ste sa vyhli oneskoreniam spôsobeným čakaním jednej úlohy na dokončenie inej.
- ✧ Plánovanie závisí od intuície a skúseností projektových manažérov.

Proces plánovania projektu



Problémy s plánovaním



- ✧ Odhadnúť náročnosť problémov a tým aj náklady na vývoj riešenia je ťažké.
- ✧ Produktivita nie je úmerná počtu ľudí pracujúcich na úlohe.
- ✧ Pridaním ľudí do neskorého projektu je to neskôr z dôvodu režijných nákladov na komunikáciu.
- ✧ Neočakávané sa vždy stane. Uvažujte aj nepredvídateľné veci pri plánovaní.

Prezentácia harmonogramu



- ✧ Grafické zobrazenie
- ✧ Zobrazuje rozdelenie projektu na úlohy. Úlohy by nemali byť príliš malé. Mali by trvať asi týždeň alebo dva.
- ✧ Na základe kalendára
 - Stĺpcové grafy sú najbežnejšie používaným znázornením plánov projektov. Zobrazujú rozvrh ako aktivity alebo zdroje v závislosti od času.
- ✧ Siete aktivít
 - Zobrazuje závislosti úloh

Projektové aktivity



- ✧ Projektové aktivity (úlohy) sú základným plánovacím prvkom. Každá aktivita má:
 - trvanie v kalendárnych dňoch alebo mesiacoch,
 - úsilie , ktoré ukazuje počet osobo-dní alebo osobo-mesiakov na dokončenie práce (PD vs MD)
 - termín , do ktorého má byť činnosť ukončená,
 - definovaný koncový bod, ktorým môže byť dokument, uskutočnenie kontrolného stretnutia, úspešné vykonanie všetkých testov atď.

Míľníky a výstupy



- ✧ Míľníky sú body v harmonograme, podľa ktorých môžete hodnotiť pokrok, napríklad odovzdanie systému na testovanie.
- ✧ Dodávky sú funkčné produkty, ktoré sa dodávajú zákazníkovi, napr. aj dokument požiadaviek na systém.

Úlohy , trvanie a závislosti



Software Engineering
6th edition

Ian Sommerville

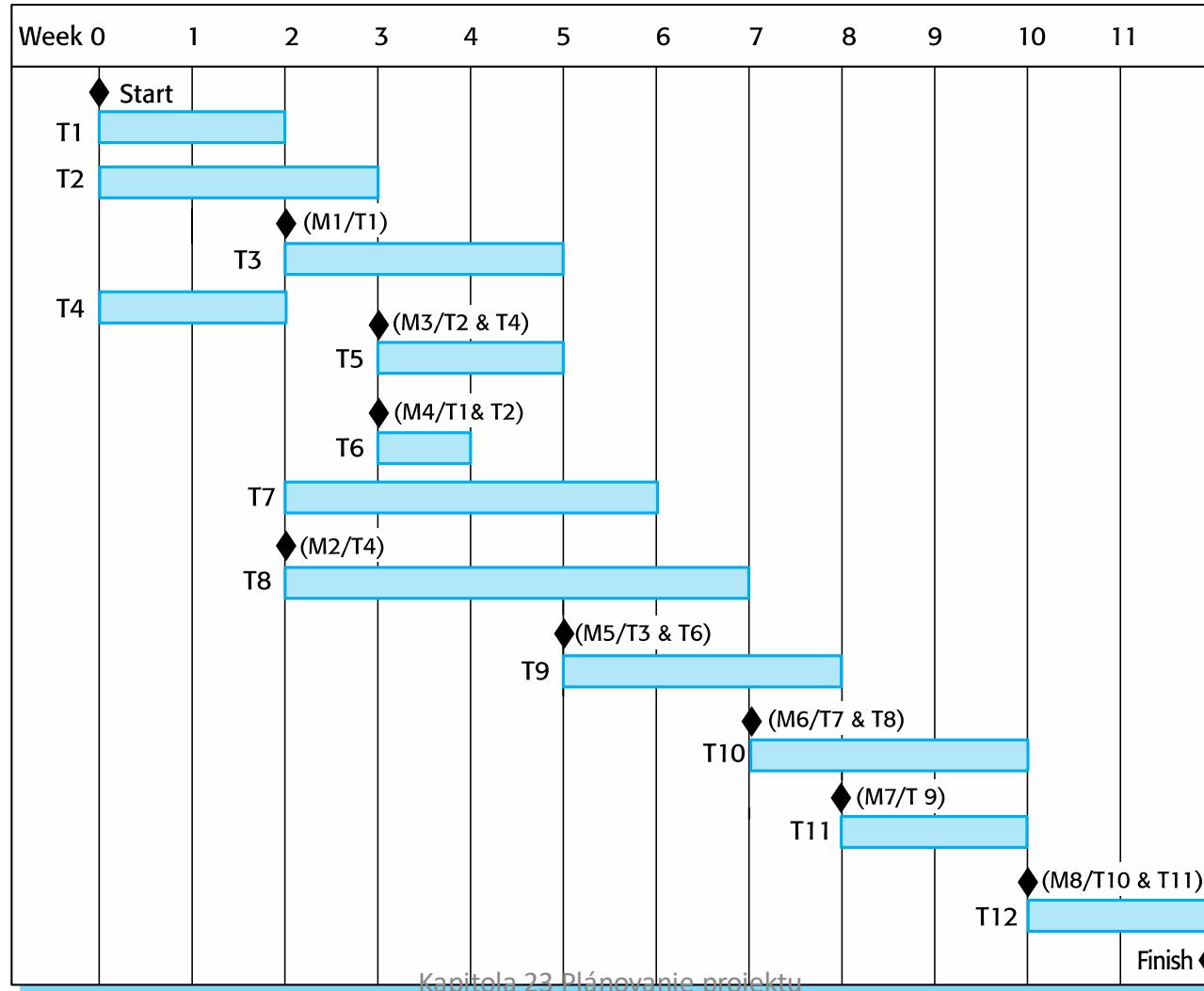
Úloha	Úsilie (osobo-dni)	Trvanie (dni)	Závislosti
T1	15	10	
T2	8	15	
T3	20	15	T1 (M1)
T4	5	10	
T5	5	10	T2, T4 (M3)
T6	10	5	T1, T2 (M4)
T7	25	20	T1 (M1)
T8	75	25	T4 (M2)
T9	10	15	T3, T6 (M5)
T10	20	15	T7, T8 (M6)
T11	10	10	T9 (M7)
T12	20	10	T10, T11 (M8)

Stípcový graf aktivity (Gantt)

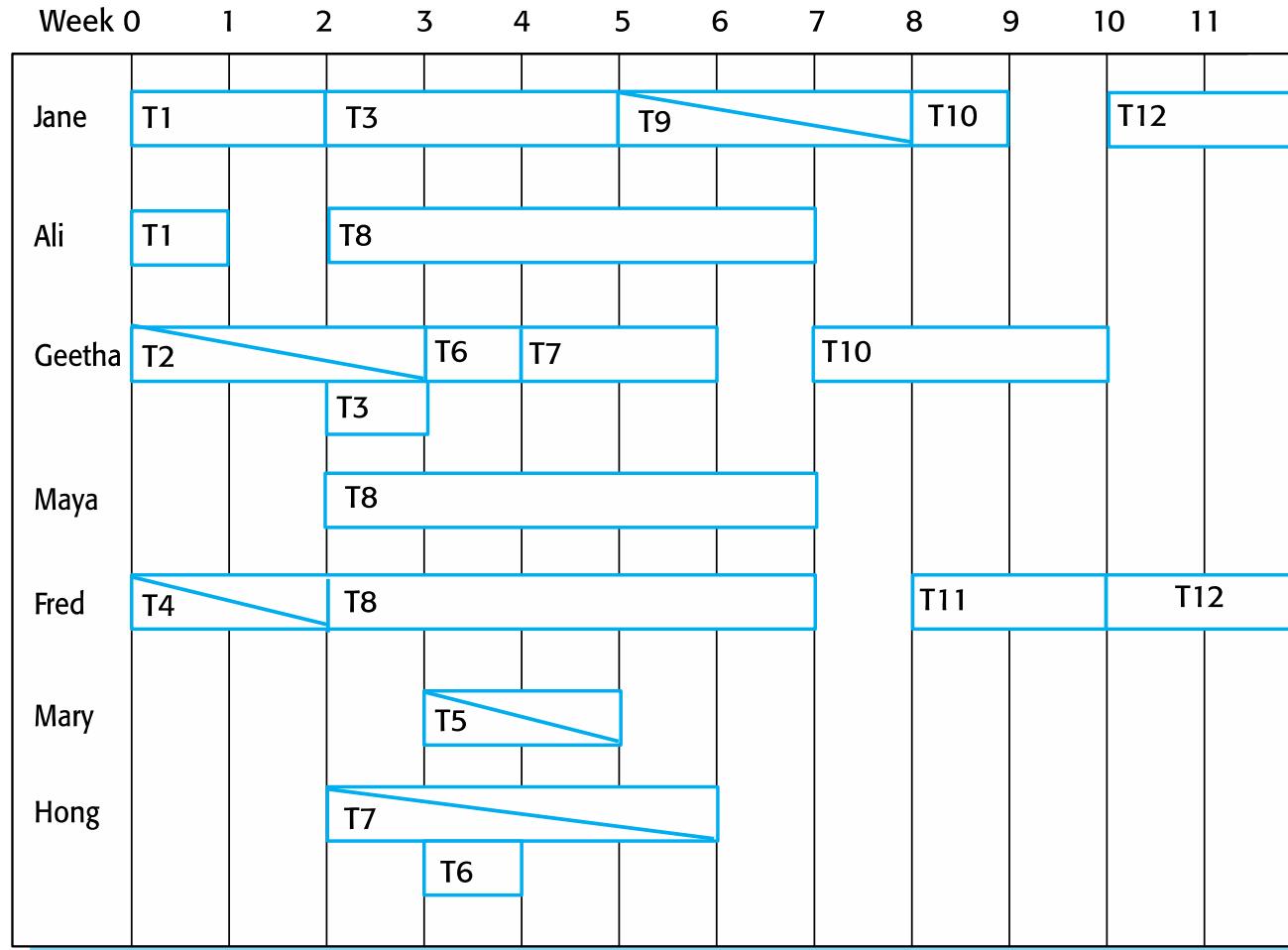


Software Engineering
6th edition

Ian Sommerville



Tabuľka pridelovania zamestnancov



Agilné plánovanie



- ✧ Agilné metódy vývoja softvéru sú iteratívne prístupy, pri ktorých sa softvér vyvíja a dodáva zákazníkom postupne.
- ✧ Na rozdiel od plánom riadených prístupov sa funkčnosť týchto prírastkov neplánuje vopred, ale rozhoduje sa o nich počas vývoja.
 - Rozhodnutie o tom, čo zahrnúť do prírastku, závisí od pokroku a od priorít zákazníka.
- ✧ Priority a požiadavky zákazníka sa menia, preto má zmysel mať flexibilný plán, ktorý sa týmto zmenám dokáže prispôsobiť.

Etapy agilného plánovania



- ✧ **Plánovanie vydania**, ktoré sa pozera dopredu na niekoľko mesiacov a rozhoduje o funkciách, ktoré by mali byť súčasťou vydania systému.
- ✧ **Iteračné plánovanie**, ktoré má krátkodobý výhľad a zameriava sa na plánovanie ďalšieho prírastku systému. Zvyčajne ide o 2-4 týždne práce pre tím.

Prístupy k agilnému plánovaniu



- ✧ Plánovanie v Scrumе
- ✧ Založené na riadení nevybavených úloh
 - Veci, ktoré treba robiť s dennými kontrolami pokroku a problémov
- ✧ Plánovacia hra
 - Pôvodne vyvinutý ako súčasť Extreme Programming (XP)
 - Závisí od príbehov používateľov ako miery pokroku v projekte

Plánovanie založené na príbehoch



- ✧ Plánovacia hra je založená na používateľských príbehoch, ktoré odrážajú funkcie, ktoré by mal systém obsahovať.
- ✧ Projektový tím si príbehy prečíta a prediskutuje a zoradí ich podľa času, ktorý si myslia, že implementácia príbehu bude trvať.
- ✧ Príbehom sú priradené „body úsilia“ odrážajúce ich veľkosť a náročnosť implementácie
- ✧ Meria sa počet bodov úsilia implementovaných za deň, čím sa získa odhad „rýchlosťi“ tímu
- ✧ To umožňuje odhadnúť celkové úsilie potrebné na implementáciu systému

Plánovacia hra





Riadenie rizík

Riadenie rizík



- ✧ Riadenie rizík sa zaoberá identifikáciou rizík a zostavovaním plánov na minimalizáciu ich vplyvu na projekt .
- ✧ Manažment softvérových rizík je dôležitý kvôli neistotám spojeným s vývojom softvéru.
 - Tieto neistoty pramenia z voľne definovaných požiadaviek, zmien požiadaviek v dôsledku zmien potrieb zákazníkov, ťažkostí pri odhadovaní času a zdrojov potrebných na vývoj softvéru a rozdielov v individuálnych zručnostiach.
- ✧ Musíte predvídať riziká, chápať vplyv týchto rizík na projekt, produkt a podnikanie a podniknúť kroky, aby ste sa týmto rizikám vyhli.



Klasifikácia rizika

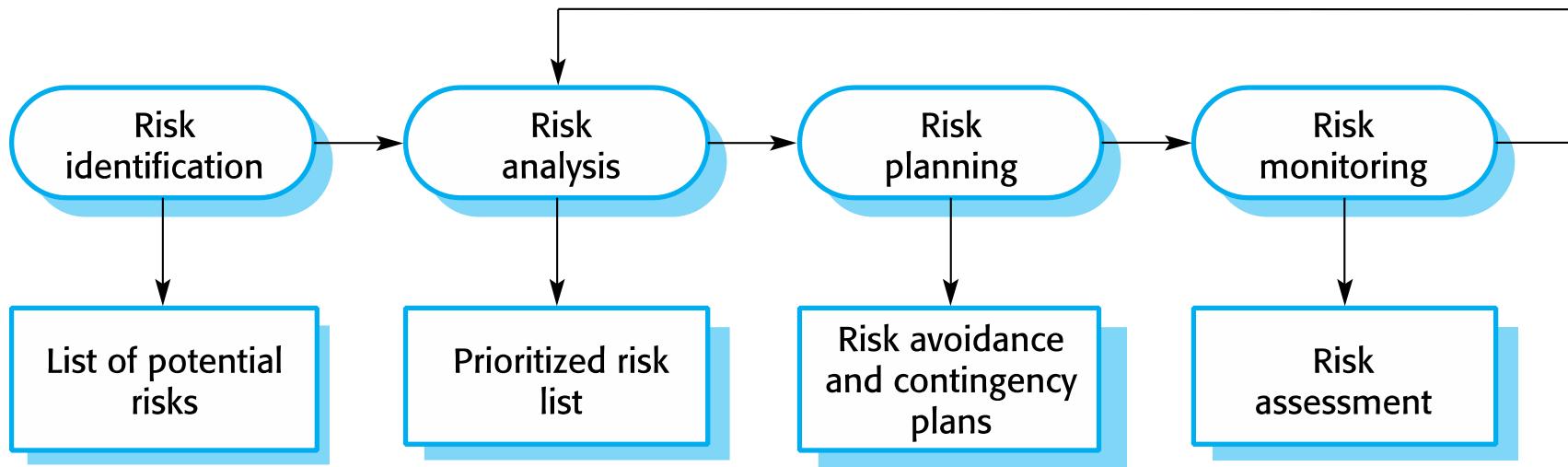
- ✧ Existujú dva rozmery klasifikácie rizika
 - Druh rizika (technické, organizačné, ...)
 - čo je ovplyvnené rizikom:
- ✧ *Riziká projektu* ovplyvniť plán alebo zdroje ;
- ✧ *Riziká produktu* ovplyvniť kvalitu alebo výkon vyvíjaného softvéru ;
- ✧ *Podnikateľské riziká* ovplyvňujú organizáciu, ktorá vyvíja alebo obstaráva softvér.

Príklady rizík projektu , produktov a podnikania



Riziko	Ovplyvňuje	Popis
Fluktuácia zamestnancov	Projekt	Skúsení pracovníci opustia projekt ešte pred jeho dokončením.
Zmena manažmentu	Projekt	Dôjde k zmene organizačného manažmentu s odlišnými prioritami.
Nedostupnosť hardvéru	Projekt	Hardvér, ktorý je pre projekt nevyhnutný, nebude dodaný podľa plánu.
Požiadavky sa menia	Projekt a produkt	V požiadavkách dôjde k väčšiemu počtu zmien, než sa predpokladalo.
Oneskorenia špecifikácií	Projekt a produkt	Špecifikácie základných rozhraní nie sú k dispozícii podľa plánu.
Veľkosť podhodnotená	Projekt a produkt	Veľkosť systému bola podcenená.
Nedostatočný výkon nástroja CASE	Produkt	CASE nástroje, ktoré podporujú projekt, nefungujú tak, ako sa očakávalo.
Zmena technológie	Podnikanie	Základná technológia, na ktorej je systém postavený, je nahradená novou technológiou.
Konkurencia produktov	Podnikanie	Konkurenčný produkt je uvedený na trh pred dokončením systému.

Proces riadenia rizík



Proces riadenia rizík



✧ Identifikácia rizika

- Identifikujte projektové, produktové a obchodné riziká;

✧ Analýza rizík

- Posúdiť pravdepodobnosť a dôsledky týchto rizík;

✧ Plánovanie rizika

- Vypracujte plány na zabránenie alebo minimalizovanie účinkov rizika;

✧ Monitorovanie rizík

- Monitorujte riziká počas celého projektu;

Identifikácia rizika



- ✧ Môže ísť o tímové aktivity alebo na základe skúseností jednotlivých projektových manažérov.
- ✧ Na identifikáciu rizík v projekte možno použiť kontrolný zoznam spoločných rizík
 - Technologické riziká .
 - Organizačné riziká .
 - Ľudské riziká.
 - Riziká z požiadaviek .
 - Riziká z odhadov .

Príklady rôznych typov rizík



Typ rizika	Možné riziká
Odhad	Čas potrebný na vývoj softvéru je podhodnotený. (12) Miera opravy defektu je podhodnotená. (13) Veľkosť softvéru je podhodnotená. (14)
Organizačné	Organizácia je reštrukturalizovaná tak, že za projekt je zodpovedný iný manažment. (6) Finančné problémy organizácie si vynucujú zníženie rozpočtu projektu. (7)
Ludia	Nie je možné získať zamestnancov s požadovanými zručnosťami. (3) Kľúčoví zamestnanci sú v kritických časoch chorí a nedostupní. (4) Požadované školenie pre zamestnancov nie je k dispozícii. (5)
Požiadavky	Navrhujú sa zmeny požiadaviek, ktoré si vyžadujú veľké prepracovanie dizajnu. (10) Zákazníci nedokážu pochopiť vplyv zmien požiadaviek. (11)
Technológia	Databáza používaná v systéme nedokáže spracovať toľko transakcií za sekundu, ako sa očakáva. (1) Opäťovne použiteľné softvérové komponenty obsahujú chyby, ktoré znamenajú, že ich nemožno opäťovne použiť podľa plánu. (2)
Nástroje	Kód generovaný nástrojmi na generovanie softvérového kódu je neefektívny. (8) Softvérové nástroje nemôžu spolupracovať integrovaným spôsobom. (9)

Analýza rizík



- ✧ Posúdťte pravdepodobnosť a závažnosť každého rizika.
- ✧ Pravdepodobnosť môže byť veľmi nízka, nízka, stredná, vysoká alebo veľmi vysoká.
- ✧ Rizikové dôsledky môžu byť katastrofálne, vážne, tolerovateľné alebo bezvýznamné.



Rizík a príklady

Riziko	Pravdepodobnosť	Účinky
organizácie si vynucujú zníženie rozpočtu projektu (7).	Nízka	Katastrofálne
Nie je možné získať zamestnancov so zručnosťami požadovanými pre projekt (3).	Vysoká	Katastrofálne
Kľúčoví zamestnanci sú chorí v kritických časoch projektu (4).	Mierne	Vážne
Chyby v opakovane použiteľných softvérových komponentoch sa musia opraviť skôr, ako sa tieto komponenty opäťovne použijú. (2).	Mierne	Vážne
Navrhujú sa zmeny požiadaviek, ktoré si vyžadujú veľké prepracovanie dizajnu (10).	Mierne	Vážne
Organizácia je reštrukturalizovaná tak, že za projekt je zodpovedný iný manažment (6).	Vysoká	Vážne
Databáza používaná v systéme nedokáže spracovať toľko transakcií za sekundu, ako sa očakáva (1). <small>Kapitola 22 Projektový manažment</small>	Mierne	Vážne

Typy rizík a príklady



Riziko	Pravdepodobnosť	Účinky
Čas potrebný na vývoj softvéru sa podceňuje (12).	Vysoká	Vážne
Softvérové nástroje nemožno integrovať (9).	Vysoká	Znesiteľné
Zákazníci nedokážu pochopiť vplyv zmien požiadaviek (11).	Mierne	Znesiteľné
Požadované školenie pre zamestnancov nie je k dispozícii (5).	Mierne	Znesiteľné
Miera opravy defektov je podhodnotená (13).	Mierne	Znesiteľné
Veľkosť softvéru je podhodnotená (14).	Vysoká	Znesiteľné
Kód generovaný nástrojmi na generovanie kódu je neefektívny (8).	Mierne	Bezvýznamný



Plánovanie rizika

- ✧ Zvážte každé riziko a vytvorte stratégiu na riadenie tohto rizika.
- ✧ **Stratégie vyhýbania sa**
 - Pravdepodobnosť vzniku rizika je znížená;
- ✧ **Stratégie minimalizácie**
 - Zníži sa vplyv rizika na projekt alebo produkt;
- ✧ **Pohotovostné plány**
 - Ak riziko vznikne, pohotovostné plány sú plány na riešenie tohto rizika;

Čo keby otázky



- ✧ Čo ak je chorých viacero inžinierov súčasne?
- ✧ Čo ak ekonomický pokles povedie k zníženiu rozpočtu projektu o 20 %?
- ✧ Čo ak výkon softvéru s otvoreným zdrojovým kódom nedostatočný a jediný odborník na tento softvér s otvoreným zdrojovým kódom odíde?
- ✧ Čo ak spoločnosť, ktorá dodáva a udržiava softvérové komponenty, zanikne?
- ✧ Čo ak zákazník nedoručí upravené požiadavky tak, ako sa predpokladalo?

Stratégie , ktoré pomáhajú riadiť riziko



Riziko	Stratégia
Organizačné finančné problémy	Pripravte inštruktážny dokument pre vrcholový manažment, ktorý ukáže, ako projekt veľmi dôležitým spôsobom prispieva k cieľom podnikania, a uvedie dôvody, prečo by škrtky v rozpočte projektu neboli nákladovo efektívne.
Problémy s náborom	Upozorniť zákazníka na možné ťažkosti a možnosť omeškania; skúmať nákupné komponenty.
Ochorenie personálu	Preorganizujte tím tak, aby sa práce viac prekrývali a ľudia si preto navzájom rozumeli.
Chybné komponenty	Vymeňte potenciálne chybné komponenty za zakúpené komponenty so známou spoľahlivosťou.
Zmeny požiadaviek	Odvodiť informácie o sledovateľnosti na posúdenie vplyvu zmeny požiadaviek; maximalizovať informácie ukryté v dizajne.

Stratégie , ktoré pomáhajú riadiť riziko



Riziko	Stratégia
Organizačná reštrukturalizácia	Pripravte informačný dokument pre vrcholový manažment, ktorý ukáže, ako projekt veľmi dôležitým spôsobom prispieva k cieľom podnikania.
Výkon databázy	Preskúmajte možnosť nákupu výkonnejšej databázy.
Podcenený čas vývoja	Preskúmajte nákupné komponenty; preskúmať použitie generátora programov .



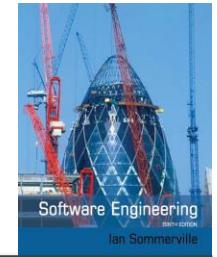
Software Engineering
6th edition

Ian Sommerville

Monitorovanie rizík

- ✧ Pravidelne vyhodnocujte každé identifikované riziko, aby ste sa rozhodli, či sa stáva menej alebo viac pravdepodobným.
- ✧ Posúdťte tiež, či sa účinky rizika zmenili.
- ✧ Každé kľúčové riziko by sa malo prediskutovať na poradách vedenia.

Ukazovatele rizika



Typ rizika	Potenciálne ukazovatele
Odhad	Nedodržanie dohodnutého harmonogramu; neodstránenie nahlásených závad .
Organizačné	Organizačné klebety; nedostatok činnosti zo strany vrcholového manažmentu.
Ludia	Zlá morálka zamestnancov; zlé vzťahy medzi členmi tímu; vysoká fluktuácia zamestnancov.
Požiadavky	Mnoho požiadaviek mení požiadavky; sťažnosti zákazníkov.
Technológia	Oneskorené dodanie hardvéru alebo podporného softvéru; veľa hlásených technologických problémov.
Nástroje	Neochota členov tímu používať nástroje; sťažnosti týkajúce sa nástrojov CASE; požiadavky na výkonnejšie pracovné stanice.



Riadenie ľudí

Riadenie ľudí



- ✧ Ľudia sú najdôležitejším aktívom organizácie.
- ✧ Úlohy manažéra sú v podstate zamerané na ľudí. Bez pochopenia ľudí bude manažment neúspešný.
- ✧ Zlé riadenie ľudí je dôležitým prispievateľom k zlyhaniu projektu.



Faktory riadenia ľudí

✧ Dôslednosť

- So všetkými členmi tímu by sa malo zaobchádzať porovnateľným spôsobom bez obľúbencov alebo diskriminácie.

✧ Rešpekt

- Rôzni členovia tímu majú rôzne zručnosti a tieto rozdiely by sa mali rešpektovať.

✧ Inklúzia

- Zapojte všetkých členov tímu a uistite sa, že sa berú do úvahy názory ľudí.

✧ Čestnosť

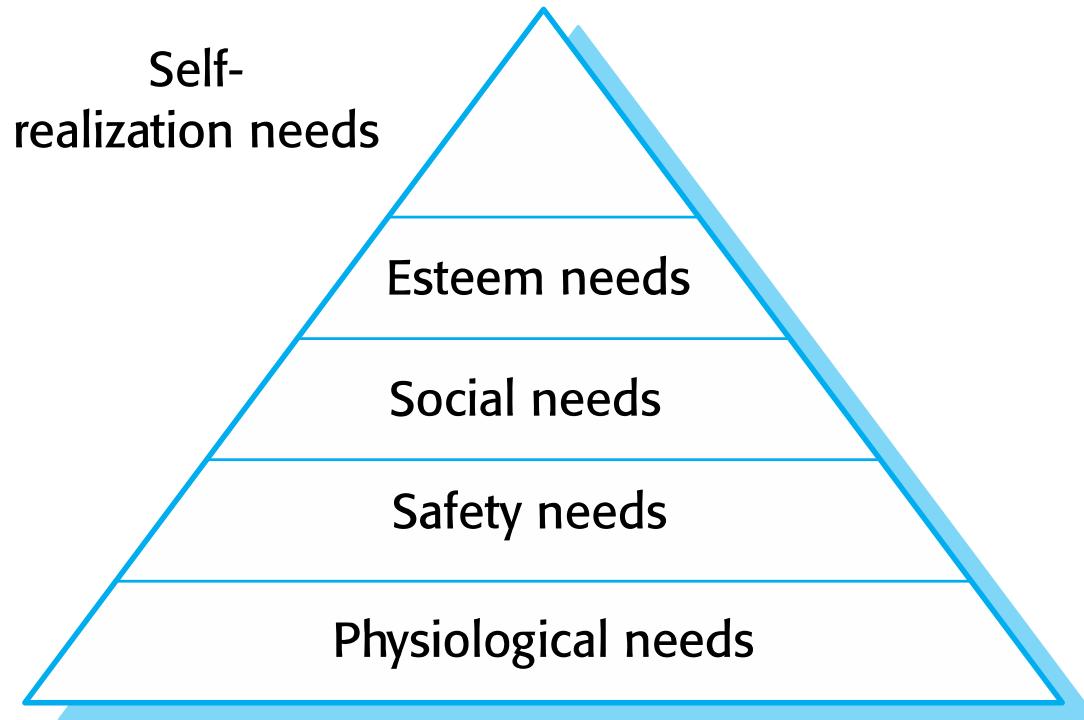
- Vždy by ste mali byť úprimní v tom, čo v projekte ide dobre a čo zle.



Motivovanie ľudí

- ✧ Dôležitou úlohou manažéra je motivovať ľudí pracujúcich na projekte .
- ✧ Motivácia znamená organizáciu práce a pracovného prostredia tak, aby povzbudzovala ľudí k efektívnej práci.
 - Ak ľudia nebudú motivovaní, práca, ktorú robia, ich nebude zaujímať. Budú pracovať pomaly, budú častejšie robiť chyby a nebudú prispievať k širším cieľom tímu alebo organizácie.
- ✧ Motivácia je zložitý problém, ale zdá sa, že existujú rôzne typy motivácie založené na:
 - Základné potreby (napr. jedlo, spánok atď.);
 - Osobné potreby (napr. rešpekt, sebaúcta);
 - Sociálne potreby (napr. byť akceptovaný ako súčasť skupiny).

Hierarchia ľudských potrieb





Potreba uspokojenia

- ✧ V skupinách vývoja softvéru nie sú základné fyziologické a bezpečnostné potreby problémom.
- ✧ Sociálna
 - Poskytovať komunálne zariadenia;
 - Umožnite neformálnu komunikáciu napr. prostredníctvom sociálnych sietí
- ✧ Úcta
 - Uznanie úspechov;
 - Primerané odmeny.
- ✧ Sebarealizácia
 - Školenie – ľudia sa chcú dozvedieť viac;
 - Zodpovednosť.

Typy osobnosti



- ✧ Hierarchia potrieb je takmer určite prílišným zjednodušením motivácie v praxi.
- ✧ Motivácia by mala brat' do úvahy aj rôzne typy osobnosti:
 - Úlohovo orientovaní ľudia, ktorí sú motivovaní prácou, ktorú robia. V softvérovom inžinierstve.
 - Orientovaný na interakciu ľudí, ktorí sú motivovaní prítomnosťou a konaním spolupracovníkov.
 - Sebaorientovaný ľudí, ktorých motivuje predovšetkým osobný úspech a uznanie.



Typy osobnosti

✧ Orientovaný na úlohy.

- Motiváciou pre vykonanie práce je samotná práca;

✧ Sebaorientovaný.

- Práca je prostriedkom na dosiahnutie cieľa, ktorým je dosiahnutie individuálnych cieľov – napr. zbohatnúť, hrať tenis, cestovať atď.;

✧ Orientovaný na interakciu

- Hlavnou motiváciou je prítomnosť a činy spolupracovníkov. Ľudia chodia do práce, pretože do práce chodia radi

Rovnováha motivácie



- ✧ Jednotlivé motivácie sa skladajú z prvkov každej triedy.
- ✧ Rovnováha sa môže meniť v závislosti od osobných okolností a vonkajších udalostí.
- ✧ Ľudia však nie sú motivovaní len osobnými faktormi, ale aj tým, že sú súčasťou skupiny a kultúry.
- ✧ Ľudia chodia do práce, pretože ich motivujú ľudia, s ktorými pracujú.



Tímová práca

Tímová práca



- ✧ Väčšina softvérového inžinierstva je skupinová činnosť
 - Plán vývoja väčšiny netriviálnych softvérových projektov je taký, že ich nemôže dokončiť jedna osoba pracujúca sama .
- ✧ Dobrá partia je súdržná a má tímového ducha.
Zainteresovaní ľudia sú motivovaní úspechom skupiny, ako aj vlastnými osobnými cieľmi.
- ✧ Skupinová interakcia je kľúčovým determinantom skupinového výkonu.
- ✧ Flexibilita v zložení skupiny je obmedzená
 - Manažéri musia robiť to najlepšie, čo môžu s dostupnými ľuďmi.



Skupinová súdržnosť

- ✧ V súdržnej skupine členovia považujú skupinu za dôležitejšiu ako ktorýkoľvek jednotlivec v nej.
- ✧ Výhody **súdržnej skupiny** sú:
 - Normy kvality skupiny môžu vytvoriť členovia skupiny.
 - tímu sa od seba navzájom učia a spoznávajú prácu toho druhého ; Znižujú sa zábrany spôsobené nevedomosťou.
 - Vedomosti sa delia. Kontinuita môže byť zachovaná, ak člen skupiny odíde.
 - Podporuje sa refactoring a neustále zlepšovanie. Členovia skupiny spoločne pracujú na poskytovaní vysokokvalitných výsledkov a riešení problémov, bez ohľadu na jednotlivcov, ktorí pôvodne vytvorili dizajn alebo program.

Efektívnosť tímu



✧ Ľudia v skupine

- V projektovej skupine potrebujete zmes ľudí, pretože vývoj softvéru zahŕňa rôzne činnosti, ako je vyjednávanie s klientmi, programovanie, testovanie a dokumentácia.

✧ Organizácia skupiny

- Skupina by mala byť organizovaná tak, aby jednotlivci mohli prispievať podľa svojich najlepších schopností a aby úlohy mohli byť splnené podľa očakávania.

✧ Technická a manažérská komunikácia

- Dobrá komunikácia medzi členmi skupiny a medzi tímom softvérového inžinierstva a ostatnými zainteresovanými stranami projektu je nevyhnutná.

Výber členov skupiny



- ✧ Úlohou manažéra alebo vedúceho tímu je vytvoriť súdržnú skupinu a organizovať svoju skupinu tak, aby mohli efektívne spolupracovať.
- ✧ To zahŕňa vytvorenie skupiny so správnou rovnováhou technických zručností a osobností a organizáciu tejto skupiny tak, aby jej členovia efektívne spolupracovali.

Zostavenie tímu



- ✧ Možno nebude možné vymenovať ideálnych ľudí na prácu na projekte
 - Rozpočet projektu nemusí umožňovať využitie vysoko platených zamestnancov;
 - Zamestnanci s príslušnými skúsenosťami nemusia byť k dispozícii;
 - Organizácia môže chcieť rozvíjať zručnosti zamestnancov na softvérovom projekte.
- ✧ Manažéri musia pracovať v rámci týchto obmedzení, najmä ak je nedostatok vyškoleného personálu.

Zloženie skupiny



- ✧ Problematická môže byť skupina zložená z členov, ktorí zdieľajú rovnakú motiváciu
 - Orientácia na úlohy – každý chce robiť to svoje;
 - Orientácia na seba – každý chce byť šéf;
 - Orientované na interakciu – príliš veľa chatovania, málo práce.
- ✧ Efektívna skupina má rovnováhu všetkých typov.
- ✧ To môže byť ťažké dosiahnuť, softvéroví inžinieri sú často orientovaní na úlohy.
- ✧ Ľudia orientovaní na interakciu sú veľmi dôležití, pretože dokážu odhaliť a zmierniť vzniknuté napätie.

Skupinová organizácia



- ✧ Spôsob, akým je skupina organizovaná, ovplyvňuje rozhodnutia, ktoré táto skupina robí, spôsoby výmeny informácií a interakcie medzi vývojovou skupinou a externými účastníkmi projektu.
 - Medzi klúčové otázky patria:
 - Mal by byť projektový manažér technickým vedúcim skupiny?
 - Kto sa bude podieľať na prijímaní dôležitých technických rozhodnutí a ako sa budú robiť?
 - Ako sa budú riešiť interakcie s externými zainteresovanými stranami a vrcholovým manažmentom spoločnosti?
 - Ako môžu skupiny integrovať ľudí, ktorí nie sú spolu?
 - Ako možno zdieľať vedomosti v rámci skupiny?

Skupinová organizácia



- ✧ **Malé skupiny** softvérového inžinierstva sú zvyčajne organizované neformálne bez pevnej štruktúry.
- ✧ Pri **veľkých projektoch** môže existovať hierarchická štruktúra, v ktorej sú rôzne skupiny zodpovedné za rôzne podprojekty .
- ✧ **Agilný vývoj** je vždy založený na neformálnej skupine na princípe, že formálna štruktúra bráni výmene informácií



Kapitola 4 – Špecifikácia požiadaviek (Inžinierstvo požiadaviek)

Inžinierstvo s požiadavkami



- ✧ Proces vytvárania **služieb**, ktoré zákazník od systému vyžaduje, a **obmedzenia**, za ktorých systém funguje a je vyvíjaný.
- ✧ Systémové požiadavky sú popisy systémových služieb a obmedzení, ktoré sú generované počas procesu navrhovania požiadaviek.

Čo je to požiadavka ?



- ✧ Môže siaháť od abstraktného vyjadrenia služby alebo systémového obmedzenia na vysokej úrovni až po podrobnú matematickú funkčnú špecifikáciu.
- ✧ Požiadavky môžu plniť dvojakú funkciu. Obe tieto vyhlásenia možno nazvať požiadavkami:
 - Môže byť základom pre ponuku na zákazku – preto musí byť otvorená pre výklad;
 - Môže byť základom pre samotnú zmluvu – preto musí byť podrobne definovaná ;

Abstrakcia požiadaviek (Davis)



„Ak chce firma uzavrieť zmluvu na veľký projekt vývoja softvéru, musí svoje potreby definovať dostatočne abstraktným spôsobom, aby riešenie nebolo vopred definované. Požiadavky musia byť napísané tak, aby sa o zákazku mohli uchádzať viacerí dodávatelia, ktorí možno ponúkajú rôzne spôsoby uspokojenia potrieb klientskej organizácie. Po zadaní zákazky musí dodávateľ napísať pre klienta definíciu systému podrobnejšie, aby klient pochopil a mohol overiť, čo softvér urobí. Oba tieto dokumenty možno nazvať dokumentom požiadaviek na systém.“

Druhy požiadaviek



✧ Požiadavky používateľov

- Príkazy v prirodzenom jazyku plus diagramy služieb, ktoré systém poskytuje, a jeho prevádzkové obmedzenia. Napísané pre zákazníkov.

✧ Požiadavky na systém

- Štruktúrovaný dokument s podrobným popisom funkcií, služieb a prevádzkových obmedzení systému. Definuje, čo sa má realizovať, takže môže byť súčasťou zmluvy medzi klientom a dodávateľom.

Používateľské a systémové požiadavky



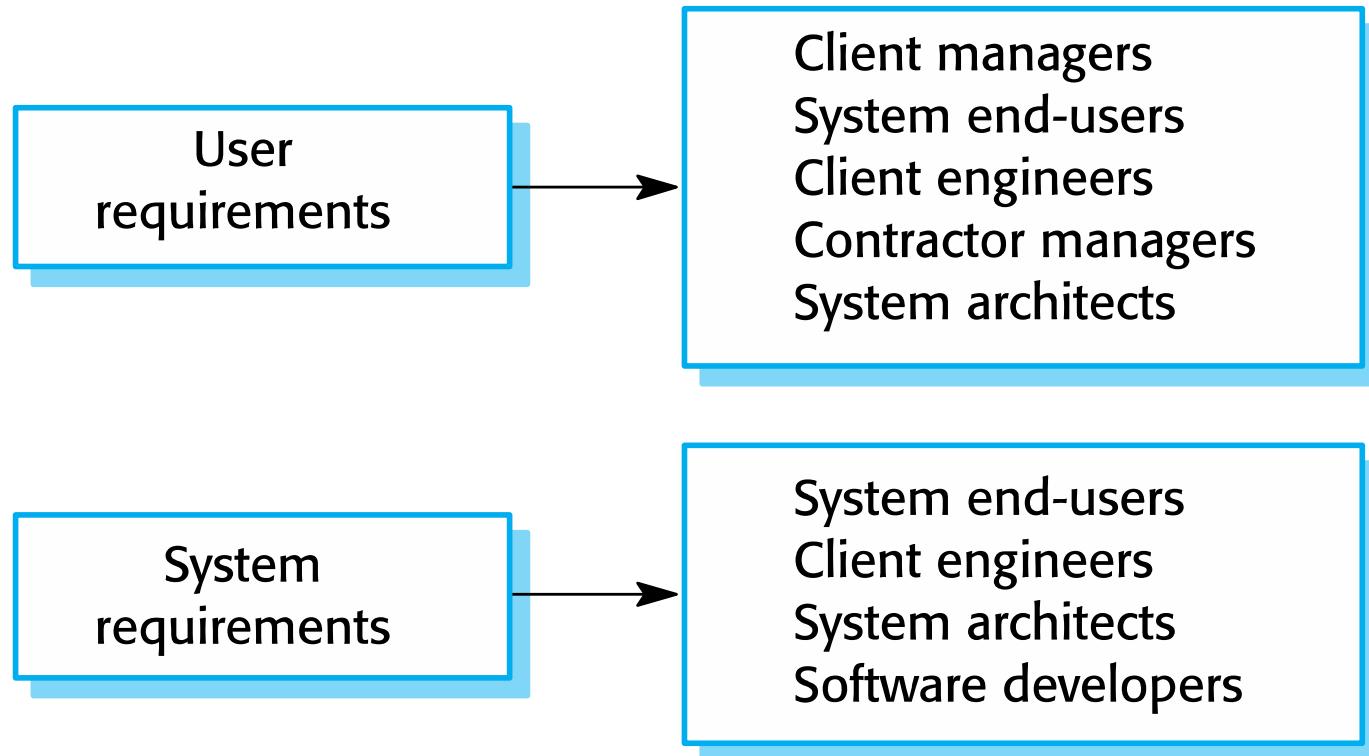
User requirements definition

- 1.** The Mentcare system shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month.

System requirements specification

- 1.1** On the last working day of each month, a summary of the drugs prescribed, their cost and the prescribing clinics shall be generated.
- 1.2** The system shall generate the report for printing after 17.30 on the last working day of the month.
- 1.3** A report shall be created for each clinic and shall list the individual drug names, the total number of prescriptions, the number of doses prescribed and the total cost of the prescribed drugs.
- 1.4** If drugs are available in different dose units (e.g. 10mg, 20mg, etc) separate reports shall be created for each dose unit.
- 1.5** Access to drug cost reports shall be restricted to authorized users as listed on a management access control list.

Ľudia na konci rôznych typov špecifikácií požiadaviek





Zainteresované strany systému

- ✧ Každá osoba alebo organizácia, ktorá je nejakým spôsobom ovplyvnená systémom a má teda oprávnený záujem
- ✧ Typy zainteresovaných strán
 - Koncoví používatelia
 - Systémoví manažéri
 - Majitelia systému
 - Externé zainteresované strany

Zainteresované strany v systéme Mentcare



- ✧ Pacienti, ktorých informácie sú zaznamenané v systéme.
- ✧ Lekári, ktorí sú zodpovední za hodnotenie a liečbu pacientov.
- ✧ Sestry, ktoré koordinujú konzultácie s lekármi a podávajú niektoré liečby.
- ✧ Recepčné ktorí riadia návštevy pacientov.
- ✧ IT pracovníci, ktorí sú zodpovední za inštaláciu a údržbu systému.

Zainteresované strany v systéme Mentcare (2)



- ✧ Manažér lekárskej etiky, ktorý musí zabezpečiť, aby systém spĺňal aktuálne etické smernice pre starostlivosť o pacienta.
- ✧ Manažéri zdravotnej starostlivosti, ktorí získavajú manažérske informácie zo systému.
- ✧ Pracovníci zdravotnej dokumentácie, ktorí sú zodpovední za zabezpečenie toho, aby sa informácie o systéme mohli udržiavať a uchovávať, a že postupy vedenia záznamov boli správne implementované.



Agilné metódy a požiadavky

- ✧ Mnohé agilné metódy tvrdia, že vytváranie podrobných systémových požiadaviek je stratou času, keďže požiadavky sa menia tak rýchlo.
- ✧ Dokument s požiadavkami je preto vždy neaktuálny.
- ✧ Agilné metódy zvyčajne využívajú inžinierstvo prírastkových požiadaviek a môžu vyjadrovať požiadavky ako „*príbehy používateľov*“
- ✧ To je praktické pre obchodné systémy, ale problematické pre systémy, ktoré vyžadujú analýzu pred dodaním (napr. kritické systémy) alebo systémy vyvinuté niekoľkými tímami.

Funkcionálne a nefunkcionálne požiadavky



✧ Funkcionálne požiadavky

- Výpisy služieb, ktoré má systém poskytovať, ako má systém reagovať na konkrétné vstupy a ako sa má správať v konkrétnych situáciach .
- Môžu uviesť, čo by systém nemal robiť.

✧ Nefunkcionálne požiadavky

- Obmedzenia služieb alebo funkcií ponúkaných systémom, ako sú časové obmedzenia, obmedzenia procesu vývoja, štandardy atď' .
- Často sa vzťahujú na systém ako celok a nie na jednotlivé funkcie alebo služby.

✧ Doménové požiadavky

- Služby a obmedzenia systému z oblasti prevádzky



Funkcionálne požiadavky

- ✧ Popisujú funkčnosť alebo systémové služby.
- ✧ Závisia od typu softvéru, očakávaných používateľov a typu systému, kde sa softvér používa.
- ✧ Funkcionálne požiadavky používateľov môžu predstavovať abstraktné vyhlásenia o tom, **čo by mal systém robiť** .
- ✧ Funkčné systémové požiadavky by mali podrobne popisovať systémové služby .

Mentcare systém: funkčné požiadavky



- ✧ Používateľ musí mať možnosť vyhľadávať v zoznamoch schôdzok pre všetky kliniky.
- ✧ Systém vygeneruje každý deň pre každú kliniku zoznam pacientov, u ktorých sa očakáva, že sa v daný deň dostavia na schôdzku.
- ✧ Každý zamestnanec používajúci systém musí byť jednoznačne identifikovaný svojím 8-miestnym číslom zamestnanca.



Nepresnosť'

- ✧ Problémy vznikajú, keď nie sú presne stanovené funkčné požiadavky.
- ✧ Nejednoznačné požiadavky môžu vývojári a používateľia interpretovať rôznymi spôsobmi.
- ✧ Zvážte výraz „hľadat“ v požiadavke č.1
 - Zámer používateľa – vyhľadávať meno pacienta na všetkých stretnutiach vo všetkých ambulanciách;
 - Výklad vývojára – hľadanie mena pacienta v jednotlivej ambulancii. Používateľ si vyberie kliniku a potom vyhľadáva.

Požiadavky na úplnosť a konzistentnosť



- ✧ V zásade by požiadavky mali byť úplné a konzistentné.
- ✧ **Úplné**
 - Mali by obsahovať popisy všetkých požadovaných zariadení.
- ✧ **Konzistentné**
 - V popisoch zariadení systému by nemali byť žiadne konflikty alebo rozpory.
- ✧ V praxi je z dôvodu systémovej a environmentálnej zložitosti nemožné vytvoriť úplný a konzistentný dokument požiadaviek.



Nefunkcionálne požiadavky

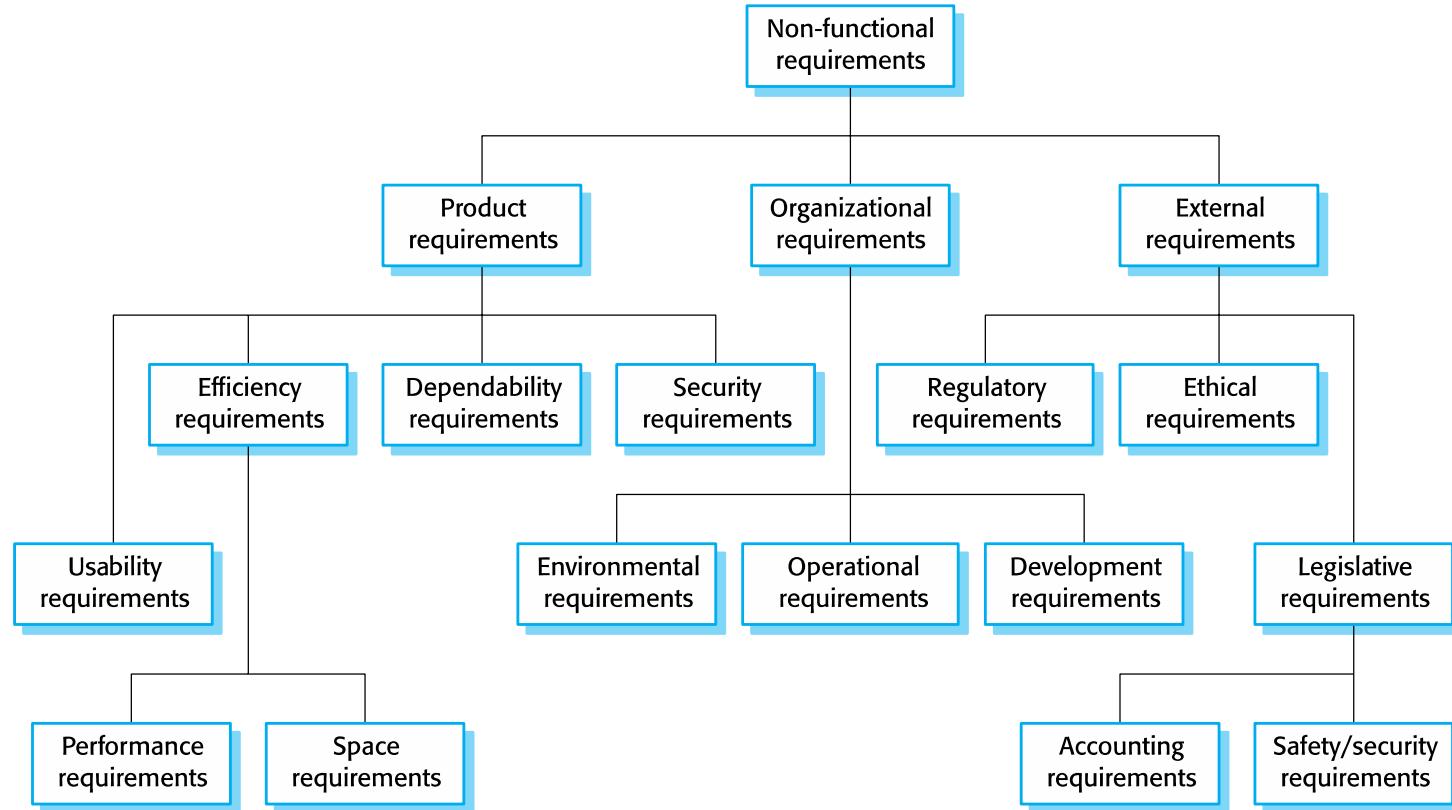
- ✧ Tieto požiadavky definujú vlastnosti systému a obmedzenia, napr. spoľahlivosť, čas odozvy a požiadavky na úložisko. Obmedzeniami sú schopnosť I/O zariadenia, reprezentácie systému atď.
- ✧ Aký by mal systém byť.
- ✧ Môžu byť špecifikované aj procesné požiadavky, ktoré vyžadujú konkrétnie IDE, programovací jazyk alebo metódu vývoja.
- ✧ Nefunkcionálne požiadavky môžu byť dôležitejšie ako funkčné požiadavky. Ak tieto nie sú splnené, systém môže byť zbytočný.



Typy nefunkcionálnych požiadaviek

Software Engineering
6th edition

Ian Sommerville



Klasifikácia nefunkcionálnych požiadaviek



✧ Požiadavky na produkt

- Požiadavky, ktoré špecifikujú, že dodaný produkt sa musí správať určitým spôsobom, napr. rýchlosť vyhotovenia, spoľahlivosť atď.

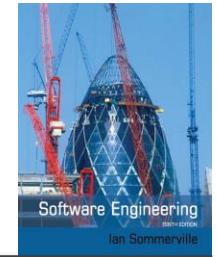
✧ Organizačné požiadavky

- Požiadavky, ktoré sú dôsledkom organizačných politík a postupov, napr. používané procesné štandardy, implementačné požiadavky atď.

✧ Externé požiadavky

- Požiadavky, ktoré vyplývajú z faktorov, ktoré sú externé pre systém a proces jeho vývoja, napr. požiadavky na interoperabilitu, legislatívne požiadavky atď.

Implementácia nefunkcionálnych požiadaviek



- ✧ Nefunkcionálne požiadavky môžu ovplyvniť skôr celkovú architektúru systému ako jednotlivé komponenty.
 - Napríklad, aby ste sa uistili, že sú splnené požiadavky na výkon, možno budete musieť zorganizovať systém tak, aby sa minimalizovala komunikácia medzi komponentmi.
- ✧ Jedna nefunkcionálna požiadavka, ako je požiadavka na bezpečnosť, môže generovať množstvo súvisiacich funkcionálnych požiadaviek, ktoré definujú systémové služby, ktoré sú požadované.
 - Môže tiež vytvárať požiadavky, ktoré obmedzujú existujúce požiadavky.

Príklady nefunkcionálnych požiadaviek v systéme Mentcare



Požiadavka na produkt

Systém Mentcare bude dostupný pre všetky kliniky počas bežnej pracovnej doby (Po – Pi, 8:30 – 17:30). Prestoje v rámci bežného pracovného času nesmú presiahnuť päť sekúnd za jeden deň.

Organizačná požiadavka

Používatelia systému Mentcare sa autentifikujú preukazom totožnosti zdravotníckeho úradu.

Vonkajšia požiadavka

Systém bude implementovať ustanovenia o ochrane súkromia pacienta, ako je uvedené v HStan-03-2006-priv.



Ciele a požiadavky

- ✧ Nefunkcionálne požiadavky môže byť veľmi ťažké presne stanoviť a nepresné požiadavky môže byť ťažké overiť.
- ✧ **Ciel'**
 - Všeobecný zámer používateľa, akým je jednoduchosť používania.
- ✧ **Overiteľná nefunkcionálna požiadavka**
 - Výrok využívajúci nejakú mieru, ktorú možno objektívne testovať (merať).
- ✧ Ciele sú užitočné pre vývojárov, pretože vyjadrujú zámery používateľov systému.

Požiadavky na použiteľnosť



- ✧ Systém by mal byť ľahko použiteľný pre zdravotnícky personál a mal by byť organizovaný tak, aby sa minimalizovali chyby používateľa. (Ciel')
- ✧ Zdravotnícky personál musí byť schopný používať všetky funkcie systému po štyroch hodinách školenia. Po tomto školení priemerný počet chýb, ktoré urobia skúsení používatelia, nepresiahne dve chyby za hodinu používania systému. (Merateľná overiteľná nefunkcionálna požiadavka)

Metriky na špecifikovanie nefunkcionálnych požiadaviek



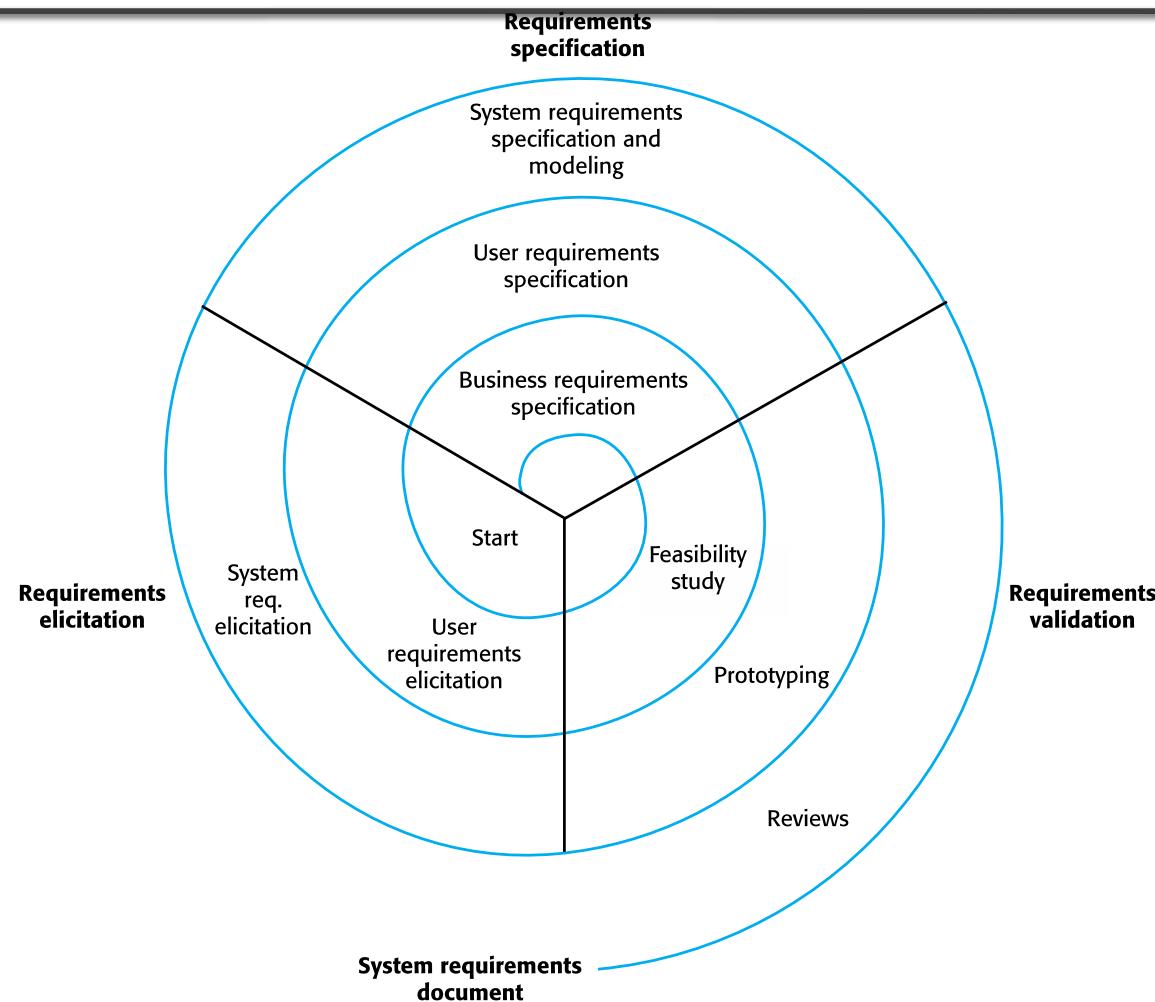
Ciel'	Miery
Rýchlosť	Spracované transakcie za sekundu Čas odozvy používateľa/udalosti Čas obnovenia obrazovky
Veľkosť	MB Počet čipov ROM
Jednoduchosť použitia	Čas na tréning Počet pomocných rámcov
Spoločalivosť	Priemerný čas do zlyhania Pravdepodobnosť nedostupnosti Miera výskytu porúch Dostupnosť
Robustnosť	Čas na reštart po zlyhaní Percento udalostí, ktoré spôsobili zlyhanie Pravdepodobnosť poškodenia údajov pri zlyhaní
Prenosnosť	Percento cieľových závislých vyhlásení Počet cieľových systémov

Inžiniersky proces na špecifikáciu požiadaviek



- ✧ Procesy používané pre špecifikáciu sa značne líšia v závislosti od aplikačnej domény, zainteresovaných ľudí a organizácií.
- ✧ Existuje však množstvo všeobecných činností spoločných pre všetky procesy
 - Získavanie požiadaviek;
 - Analýza požiadaviek;
 - Validácia požiadaviek;
 - Riadenie požiadaviek.
- ✧ V praxi je špecifikácia iteratívna činnosť, v ktorej sa tieto procesy prelínajú.

Špirálový model na proces inžinierstva požiadaviek (špecifikácie)



Získavanie a analýza požiadaviek



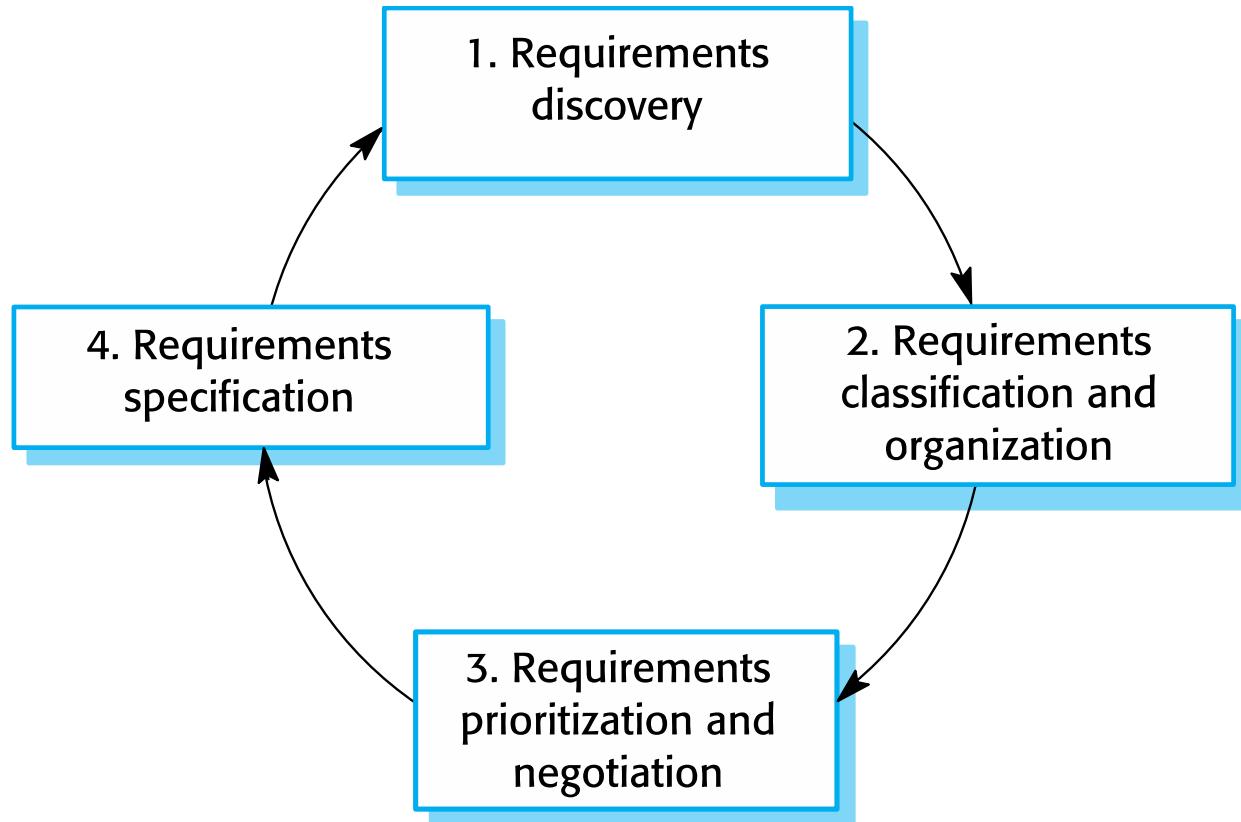
- ✧ Niekedy sa nazýva získavanie požiadaviek alebo zisťovanie požiadaviek.
- ✧ Zahŕňa technický personál pracujúci so zákazníkmi s cieľom zistiť oblast aplikácie, služby, ktoré by mal systém poskytovať, a prevádzkové obmedzenia systému.
- ✧ Môže zahŕňať koncových používateľov, manažérov, inžinierov zapojených do údržby, doménových expertov, odborové zväzy, atď .

Získavanie požiadaviek



- ✧ Softwaroví inžinieri spolupracujú s celým radom zainteresovaných strán, aby zistili aplikačnú doménu, služby, ktoré by mal systém poskytovať, požadovaný výkon systému, hardvérové obmedzenia, iné systémy atď.
- ✧ Etapy zahŕňajú:
 - Zisťovanie požiadaviek,
 - Klasifikácia a organizácia požiadaviek,
 - Stanovenie priorít a vyjednávanie požiadaviek,
 - Špecifikácia požiadaviek.

Proces získavania a analýzy požiadaviek





Procesné činnosti

✧ Zisťovanie požiadaviek

- Interakcia so zainteresovanými stranami s cieľom zistíť ich požiadavky. V tejto fáze sa zisťujú aj požiadavky na doménu.

✧ Klasifikácia a organizácia požiadaviek

- Zoskupuje súvisiace požiadavky a organizuje ich do koherentných zhľukov.

✧ Stanovenie priorít a vyjednávanie

- Uprednostňovanie požiadaviek a riešenie konfliktov požiadaviek.

✧ Špecifikácia požiadaviek

- Požiadavky sú zdokumentované a vložené do ďalšieho kola špirály.



Problémy získavania požiadaviek

- ✧ Zainteresované strany nevedia, čo vlastne chcú.
- ✧ Zainteresované strany vyjadrujú požiadavky vo svojich vlastných podmienkach.
- ✧ Rôzne zainteresované strany môžu mať protichodné požiadavky.
- ✧ Organizačné a politické faktory môžu ovplyvniť systémové požiadavky.
- ✧ Požiadavky sa počas procesu analýzy menia. Môžu sa objaviť noví zainteresovaní a podnikateľské prostredie sa môže zmeniť .

Zistovanie požiadaviek



- ✧ Proces zhromažďovania informácií o požadovaných a existujúcich systémoch a filtrovanie užívateľských a systémových požiadaviek z týchto informácií.
- ✧ Interakcia prebieha so zainteresovanými stranami systému od manažérov až po externých regulátorov.
- ✧ Systémy majú zvyčajne celý rad zainteresovaných strán.

Rozhovor



- ✧ Formálne alebo neformálne rozhovory so zainteresovanými stranami sú súčasťou väčšiny procesov špecifikácie
- ✧ Typy rozhovoru
 - Uzavreté rozhovory na základe vopred stanoveného zoznamu otázok
 - Otvorené rozhovory, kde sa skúmajú rôzne problémy so zainteresovanými stranami.
- ✧ Efektívny rozhovor
 - Budťte otvorení, vyhnite sa vopred vytvoreným predstavám o požiadavkách a budťte ochotní počúvať zainteresované strany.
 - Vyzvite účastníka rozhovoru, aby rozbehol diskusiu pomocou otázky odrazového mostíka, návrhu požiadaviek alebo spoluprácou na prototype systému.

Rozhovory v praxi



- ✧ Zvyčajne ide o kombináciu uzavretých a otvorených rozhovorov.
- ✧ Rozhovory sú dobré na získanie celkového pochopenia toho, čo zainteresované strany robia a ako môžu interagovať so systémom .
- ✧ Anketári musia byť otvorení bez vopred vytvorených predstáv o tom, čo by mal systém robiť
- ✧ Musíte podnietiť používateľov, aby hovorili o systéme tým, že navrhnete požiadavky, a nie jednoducho sa ich spýtate, čo chcú.



Problémy s pochovormi

- ✧ Aplikační špecialisti môžu na opis svojej práce používať jazyk, ktorý nie je pre inžiniera požiadaviek ľahké pochopit'.
- ✧ Rozhovory nie sú dobré na pochopenie doménových požiadaviek
 - Inžinieri požiadaviek nerozumejú špecifickej doménovej terminológií;
 - Niektoré znalosti domény sú také známe, že je pre ľudí ľažké ich formulovať alebo si myslia, že to nestojí za to formulovať ich.

Etnografia



- ✧ Sociálny vedec trávi značný čas pozorovaním a analýzou toho, ako ľudia skutočne fungujú.
- ✧ Ľudia nemusia svoju prácu vysvetľovať ani formulovať.
- ✧ Je možné pozorovať dôležité sociálne a organizačné faktory.
- ✧ Etnografické štúdie ukázali, že práca je zvyčajne bohatšia a zložitejšia, ako naznačujú jednoduché modely systémov.

Rozsah etnografie



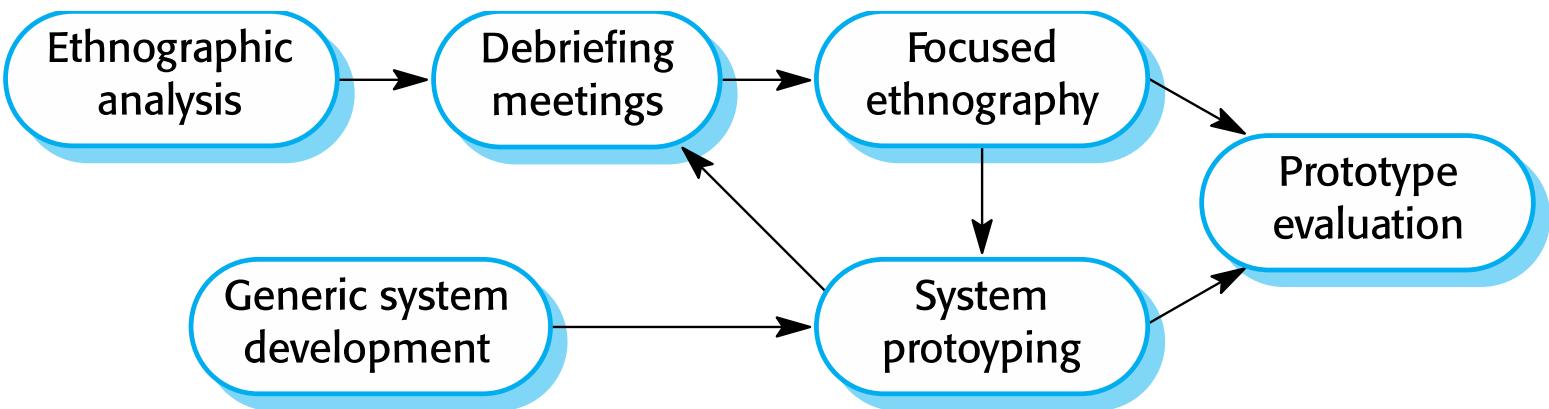
- ✧ Požiadavky, ktoré sú odvodené zo spôsobu, akým ľudia skutočne pracujú, a nie zo spôsobu, ktorý definície procesov naznačujú, že by mali pracovať.
- ✧ Požiadavky, ktoré sú odvodené od spolupráce a uvedomenia si aktivít iných ľudí .
 - Uvedomenie si toho, čo robia iní ľudia, vedie k zmenám v spôsoboch, akými robíme veci.
- ✧ Etnografia je účinná na pochopenie existujúcich procesov, ale nedokáže identifikovať nové funkcie, ktoré by mali byť pridané do systému.

Zameraná etnografia



- ✧ Vyvinuté v projekte študujúcim proces riadenia letovej prevádzky
- ✧ Spája etnografiu s prototypovaním
- ✧ Výsledkom vývoja prototypu sú nezodpovedané otázky, na ktoré sa zameriava etnografická analýza.
- ✧ Problém etnografie je v tom, že študuje existujúce postupy, ktoré môžu mať nejaký historický základ, ktorý už nie je relevantný.

Etnografia a prototypovanie pre analýzu požiadaviek



Príbehy a scenáre



- ✧ Scenáre a príbehy používateľov sú skutočnými príkladmi toho, ako možno systém použiť.
- ✧ Príbehy a scenáre sú popisom toho, ako možno systém použiť na konkrétnu úlohu.
- ✧ Keďže sú založené na praktickej situácii, zainteresované strany sa k nim môžu vzťahovať a môžu komentovať svoju situáciu s ohľadom na príbeh.

Scenáre



- ✧ Štruktúrovaná forma používateľského príbehu
- ✧ Scenáre by mali zahŕňať
 - Opis východiskovej situácie;
 - Popis normálneho toku udalostí;
 - Popis toho, čo sa môže pokaziť;
 - Informácie o iných súbežných činnostiach;
 - Popis stavu, keď sa scenár skončí.

Špecifikácia požiadaviek



- ✧ Proces zapisovania používateľských a systémových požiadaviek do dokumentu požiadaviek.
- ✧ Požiadavky používateľov musia byť zrozumiteľné pre koncových používateľov a zákazníkov, ktorí nemajú technické vzdelanie.
- ✧ Systémové požiadavky sú podrobnejšie požiadavky a môžu obsahovať viac technických informácií.
- ✧ Požiadavky môžu byť súčasťou zmluvy na vývoj systému
 - Preto je dôležité, aby boli čo najkompletnejšie.

Spôsoby písania špecifikácie systémových požiadaviek



Notový zápis	Popis
Prirodzený jazyk	Požiadavky sú napísané pomocou očíslovaných viet v prirodzenom jazyku. Každá veta by mala vyjadrovať jednu požiadavku.
Štruktúrovaný prirodzený jazyk	Požiadavky sú napísané v prirodzenom jazyku na štandardnom formulári alebo šablóne. Každé pole poskytuje informácie o aspekte požiadavky.
Špecifikačné a návrhové jazyky	Tento prístup používa jazyk ako programovací jazyk, ale s abstraktnejšími funkciami na špecifikáciu požiadaviek definovaním operačného modelu systému. Tento prístup sa v súčasnosti používa zriedka, hoci môže byť užitočný pre špecifikácie rozhrania.
Grafické záписy	Na definovanie funkčných požiadaviek na systém sa používajú grafické modely doplnené textovými anotáciami; Bežne sa používajú diagramy prípadov použitia UML a sekvenčné diagramy.
Matematické špecifikácie	Tieto záписy sú založené na matematických konceptoch, ako sú konečné stroje alebo množiny. Hoci tieto jednoznačné špecifikácie môžu znížiť nejednoznačnosť v dokumente požiadaviek, väčšina zákazníkov nerozumie formálnej špecifikácii. Nemôžu skontrolovať, či predstavuje to, čo chcú, a zdráhajú sa to priať ako systémovú zmluvu



Požiadavky a dizajn

- ✧ V zásade by **požiadavky** mali uvádzat', **čo** má systém robiť a **aký** má byť, a **návrh** by mal popisovať', **ako** to systém má robiť'.
- ✧ V praxi sú požiadavky a dizajn neoddeliteľné
 - Architektúra systému môže byť navrhnutá tak, aby štruktúrovala požiadavky;
 - Systém môže spolupracovať s inými systémami, ktoré vytvárajú konštrukčné požiadavky;
 - Použitie špecifickej architektúry na uspokojenie nefunkčných požiadaviek môže byť doménovou požiadavkou .
 - Môže to byť dôsledok regulačnej požiadavky.

Odporúčania pre písanie požiadaviek



- ✧ Vymyslite štandardný formát a použite ho pre všetky požiadavky.
- ✧ Používajte jazyk konzistentným spôsobom. Používa sa pre povinné požiadavky, malo by sa používať pre požadované požiadavky.
- ✧ Pomocou zvýraznenia textu identifikujte kľúčové časti požiadavky.
- ✧ Vyhnite sa používaniu počítačového žargónu .
- ✧ Zahrňte vysvetlenie (dôvod), prečo je požiadavka potrebná.



Problémy s prirodzeným jazykom

✧ Nejasnosť

- Presnosť je ľažká bez toho, aby bol dokument ľažko čitateľný.

✧ Zmätok požiadaviek

- Funkčné a nefunkčné požiadavky bývajú zamieňané.

✧ Zlúčenie požiadaviek

- Spolu môže byť vyjadrených niekoľko rôznych požiadaviek.

Príklad požiadaviek na softvérový systém inzulínovej pumpy



3.2 Systém meria hladinu cukru v krvi a podáva inzulín, ak je to potrebné, každých 10 minút. (*Zmeny hladiny cukru v krvi sú relatívne pomalé, takže častejšie meranie nie je potrebné; menej časté meranie by mohlo viest k zbytočne vysokej hladine cukru.*)

3.6 Systém musí každú minútu spustiť samočinný test s podmienkami, ktoré sa majú testovať, a súvisiacimi činnosťami definovanými v tabuľke 1. (*Samočinný test môže odhaliť hardvérové a softvérové problémy a upozorniť používateľa na skutočnosť, že normálna prevádzka môže byť nemožné.*)

Štruktúrované špecifikácie



- ✧ Prístup k písaniu požiadaviek, kde je obmedzená sloboda autora požiadaviek a požiadavky sú písané štandardným spôsobom.
- ✧ Toto funguje dobre pre niektoré typy požiadaviek, napr. požiadavky na vstavaný riadiaci systém, ale niekedy je to príliš rigidné na písanie požiadaviek na obchodný systém.

Špecifikácie založené na formulároch



- ✧ Definícia funkcie alebo entity.
- ✧ Popis vstupov a odkiaľ pochádzajú.
- ✧ Popis výstupov a kam smerujú.
- ✧ Informácie o informáciách potrebných na výpočet a iných použitých entitách.
- ✧ Opis akcie, ktorá sa má vykonať.
- ✧ Podmienky pred a po (ak je to vhodné).
- ✧ Vedľajšie účinky (ak existujú).

Štruktúrovaná špecifikácia požiadavky na inzulínovú pumpu



Insulin Pump/Control Software/SRS/3.3.2

Function Compute insulin dose: safe sugar level.

Description

Computes the dose of insulin to be delivered when the current measured sugar level is in the safe zone between 3 and 7 units.

Inputs Current sugar reading (r_2); the previous two readings (r_0 and r_1).

Source Current sugar reading from sensor. Other readings from memory.

Outputs CompDose—the dose in insulin to be delivered.

Destination Main control loop.

Štruktúrovaná špecifikácia požiadavky na inzulínovú pumpu



Action

CompDose is zero if the sugar level is stable or falling or if the level is increasing but the rate of increase is decreasing. If the level is increasing and the rate of increase is increasing, then CompDose is computed by dividing the difference between the current sugar level and the previous level by 4 and rounding the result. If the result, is rounded to zero then CompDose is set to the minimum dose that can be delivered.

Requirements

Two previous readings so that the rate of change of sugar level can be computed.

Pre-condition

The insulin reservoir contains at least the maximum allowed single dose of insulin.

Post-condition r0 is replaced by r1 then r1 is replaced by r2.

Side effects None.

Tabuľková špecifikácia



- ✧ Používa sa na doplnenie prirodzeného jazyka.
- ✧ Obzvlášť užitočné, keď musíte definovať niekoľko možných alternatívnych postupov .
- ✧ Napríklad systémy inzulínovej pumpy zakladajú svoje výpočty na rýchlosť zmeny hladiny cukru v krvi a tabuľková špecifikácia vysvetľuje, ako vypočítat' potrebu inzulínu pre rôzne scenáre.

Tabuľková špecifikácia výpočtu pre inzulínovú pumpu



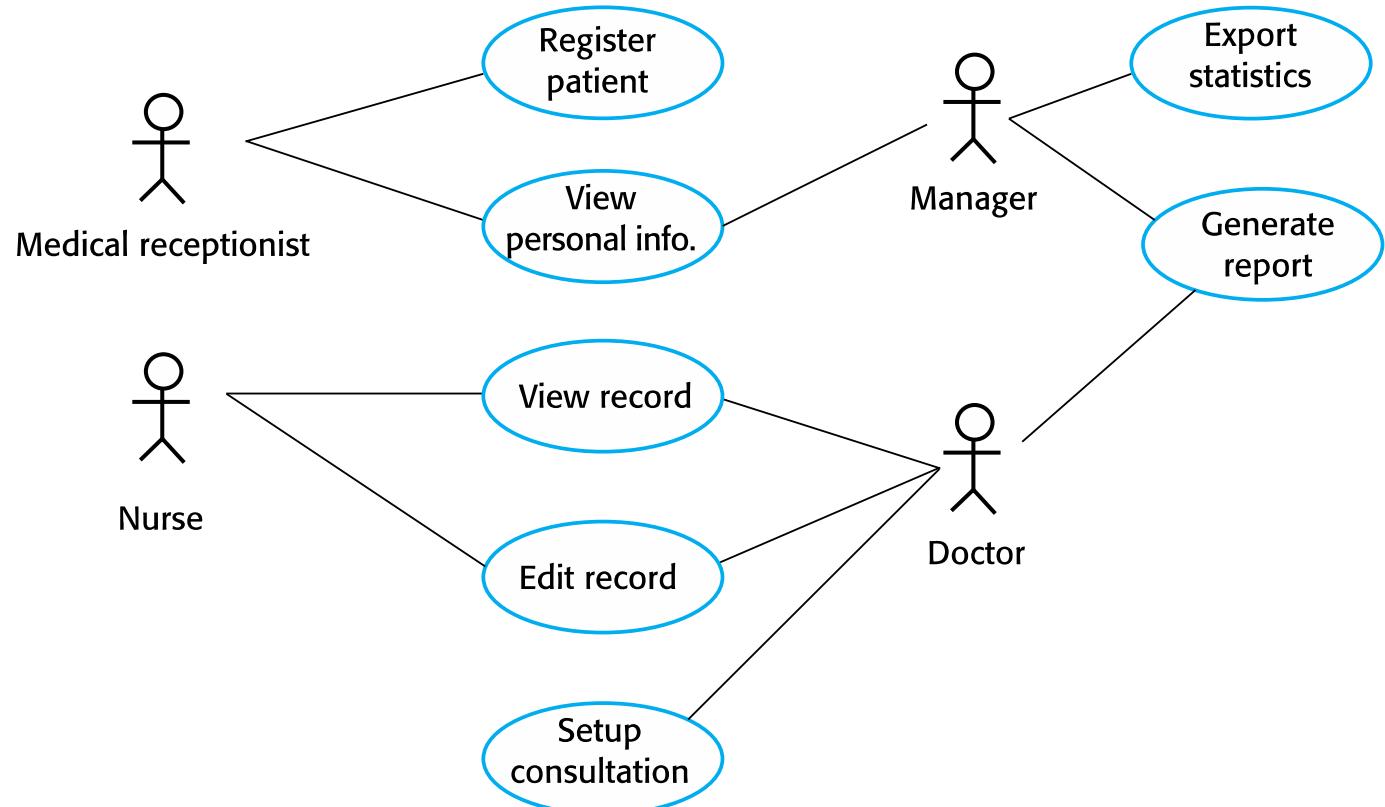
Podmienka	Akcia
Hladina cukru klesá ($r_2 < r_1$)	CompDose = 0
Stabilná hladina cukru ($r_2 = r_1$)	CompDose = 0
Hladina cukru sa zvyšuje a rýchlosť nárastu sa znižuje $((r_2 - r_1) < (r_1 - r_0))$	CompDose = 0
Zvyšovanie hladiny cukru a rýchlosť zvyšovania stabilná alebo stúpajúca $((r_2 - r_1) \geq (r_1 - r_0))$	CompDose = kolo $((r_2 - r_1)/4)$ Ak je zaokrúhlený výsledok = 0, potom CompDose = minimálna dávka



Prípady použitia

- ✧ Prípady použitia sú akýmsi scenárom, ktorý je súčasťou UML .
- ✧ Prípady použitia identifikujú aktérov interakcie a popisujú samotnú interakciu.
- ✧ Súbor prípadov použitia by mal popisovať všetky možné interakcie so systémom .
- ✧ Vysokoúrovňový grafický model doplnený o podrobnejší tabuľkový popis (pozri kapitolu 5).
- ✧ Sekvenčné diagramy UML sa môžu použiť na pridanie detailov k prípadom použitia zobrazením postupnosti spracovania udalostí v systéme.

Prípady použitia pre systém Mentcare



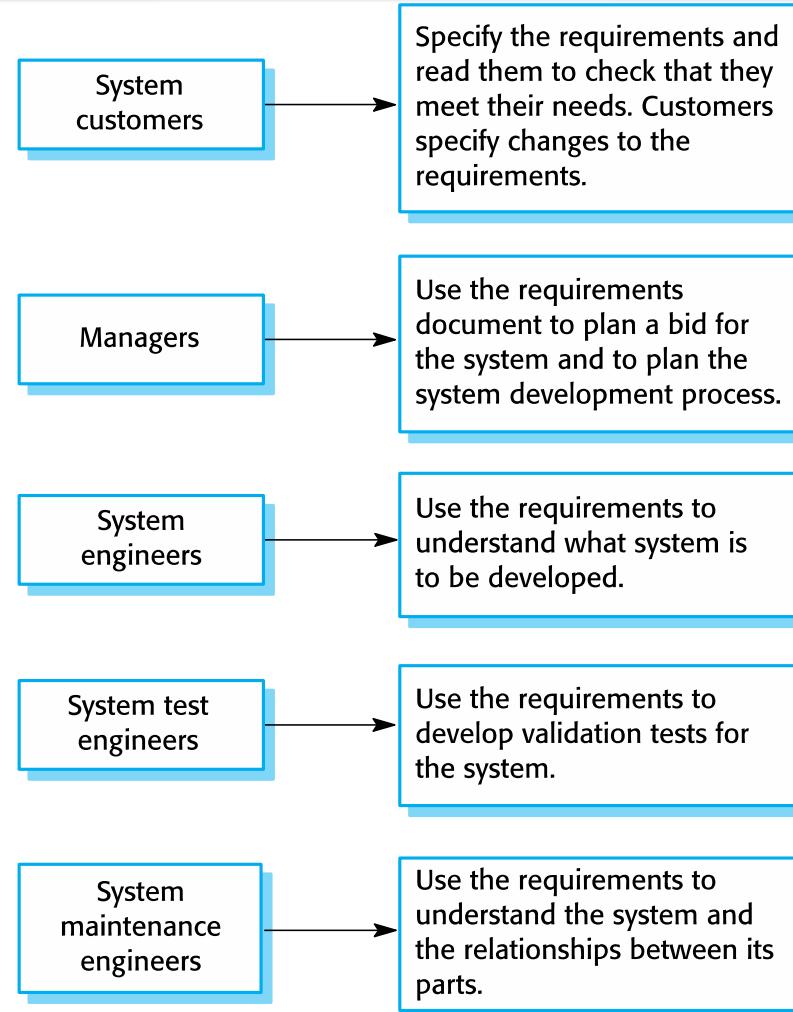


Dokument s požiadavkami na softvér

- ✧ Dokument s požiadavkami na softvér je oficiálnym vyhlásením toho, čo sa vyžaduje od vývojárov systému.
- ✧ Mal by obsahovať definíciu požiadaviek užívateľa a špecifikáciu systémových požiadaviek.
- ✧ NIE JE to dizajnový dokument. Pokiaľ je to možné, **mal by stanoviť**, **ČO** by mal systém robiť, a nie **AKO** by to mal robiť.



Používatelia dokumentu s požiadavkami



Požiadavky dokumentujú variabilitu



- ✧ Informácie v dokumente požiadaviek závisia od typu systému a použitého prístupu k vývoju.
- ✧ Systémy vyvíjané postupne budú mať zvyčajne menej podrobností v dokumente požiadaviek.
- ✧ Boli navrhnuté štandardy dokumentov požiadaviek, napr. štandard IEEE. Tieto sú väčšinou použiteľné pre požiadavky veľkých projektov systémového inžinierstva.

Overenie požiadaviek



- ✧ Zaoberá sa preukázaním, že požiadavky definujú systém, ktorý zákazník skutočne chce.
- ✧ Náklady na chyby v požiadavkách sú vysoké, takže validácia je veľmi dôležitá
 - Oprava chyby požiadaviek po doručení môže stáť až 100-násobok nákladov na opravu chyby implementácie.

Kontrola požiadaviek



- ✧ Platnosť. Poskytuje systém funkcie, ktoré najlepšie podporujú potreby zákazníka?
- ✧ Dôslednosť . Existujú nejaké konflikty požiadaviek?
- ✧ Úplnosť. Sú zahrnuté všetky funkcie požadované zákazníkom?
- ✧ Realizmus. Môžu byť požiadavky implementované vzhľadom na dostupný rozpočet a technológiu?
- ✧ Overiteľnosť. Je možné skontrolovať požiadavky?

Techniky validácie požiadaviek



✧ Recenzie požiadaviek

- Systematická manuálna analýza požiadaviek.

✧ Prototypovanie

- Použitie spustiteľného modelu systému na kontrolu požiadaviek.
Popísané v kapitole 2.

✧ Generovanie testovacích prípadov

- Vývoj testov pre požiadavky na kontrolu testovateľnosti.



Recenzie požiadaviek

- ✧ Počas formulovania definície požiadaviek by sa mali vykonávať pravidelné kontroly.
- ✧ Do kontrol by sa mali zapojiť zamestnanci klienta aj dodávateľa.
- ✧ Preskúmania môžu byť formálne (s vyplnenými dokumentmi) alebo neformálne. Dobrá komunikácia medzi vývojármi, zákazníkmi a používateľmi môže vyriešiť problémy v počiatočnom štádiu.



Kontrolné kontroly

✧ Overiteľnosť

- Je požiadavka reálne testovateľná?

✧ Zrozumiteľnosť

- Je požiadavka správne pochopená?

✧ Vysledovateľnosť

- Je jasne uvedený pôvod požiadavky?

✧ Prispôsobivosť

- Dá sa požiadavka zmeniť bez veľkého vplyvu na ostatné požiadavky?



Meniace sa požiadavky

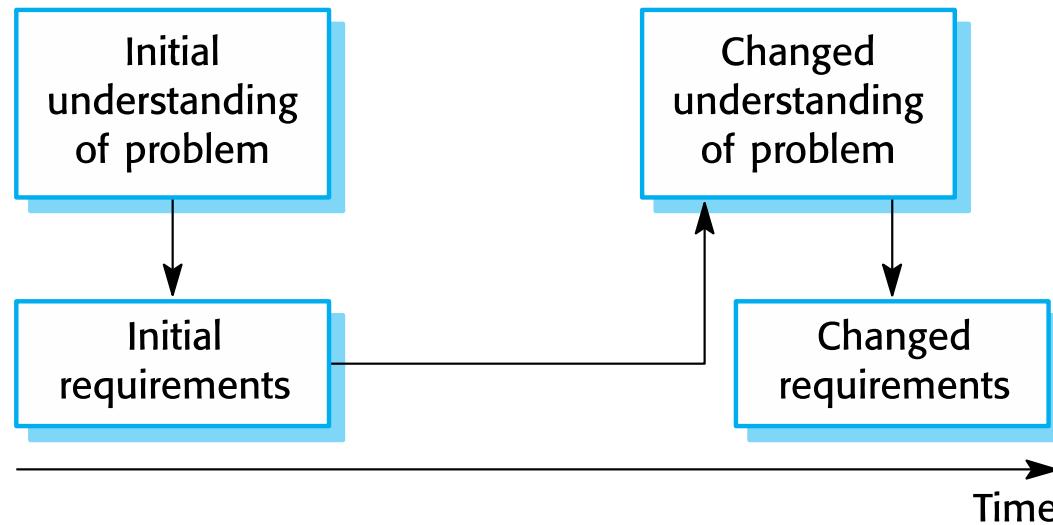
- ✧ Obchodné a technické prostredie systému sa vždy po čase mení.
 - Môže sa zaviesť nový hardvér, môže byť potrebné prepojenie systému s inými systémami, môžu sa zmeniť obchodné priority (s následnými zmenami v potrebnej podpore systému) a môžu sa zaviesť nové právne predpisy a predpisy, ktoré musí systém nevyhnutne dodržiavať.
- ✧ Ľudia, ktorí platia za systém, a používatelia tohto systému sú len zriedka tí istí ľudia.
 - Zákazníci systému kladú požiadavky z dôvodu organizačných a rozpočtových obmedzení. Tieto môžu byť v rozpore s požiadavkami koncových používateľov a po dodaní môže byť potrebné pridať nové funkcie na podporu používateľov, ak má systém splniť svoje ciele.



Zmena požiadaviek

- ✧ Veľké systémy majú zvyčajne rôznorodú komunitu používateľov, pričom mnohí používatelia majú rôzne požiadavky a priority, ktoré môžu byť protichodné.
 - Konečné systémové požiadavky sú nevyhnutne kompromisom medzi nimi a na základe skúseností sa často zistí, že vyváženie podpory poskytovanej rôznym používateľom sa musí zmeniť.

Vývoj požiadaviek



Riadenie požiadaviek



- ✧ Riadenie požiadaviek je proces riadenia meniacich sa požiadaviek počas procesu inžinierstva požiadaviek a vývoja systému .
- ✧ Nové požiadavky vznikajú pri vývoji systému a po jeho uvedení do prevádzky.
- ✧ Musíte sledovať jednotlivé požiadavky a udržiavať prepojenia medzi závislými požiadavkami, aby ste mohli posúdiť vplyv zmien požiadaviek. Musíte vytvoriť formálny proces na vytváranie návrhov zmien a ich prepojenie so systémovými požiadavkami.



Plánovanie manažmentu požiadaviek

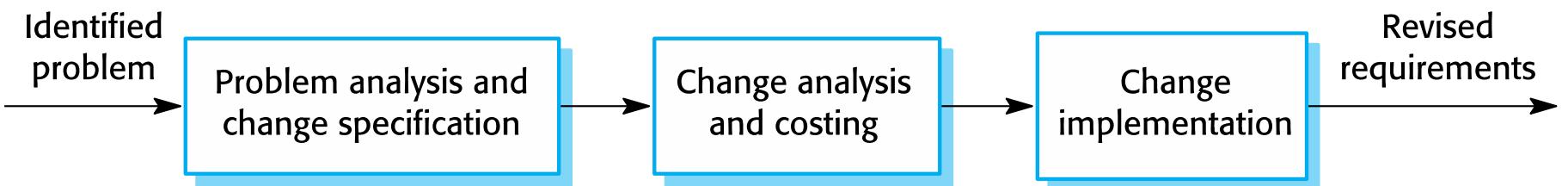
- ✧ Stanovuje úroveň podrobnosti o správe požiadaviek, ktorá sa vyžaduje.
- ✧ Rozhodnutia manažmentu požiadaviek:
 - *Identifikácia požiadaviek* Každá požiadavka musí byť jednoznačne identifikovaná, aby sa dala porovnať s inými požiadavkami.
 - *Proces riadenia zmien* Ide o súbor činností, ktoré hodnotia vplyv a náklady zmien. Podrobnejšie sa tomuto procesu venujem v nasledujúcej časti.
 - *Politiky sledovateľnosti* Tieto zásady definujú vzťahy medzi každou požiadavkou a medzi požiadavkami a návrhom systému, ktorý by sa mal zaznamenať.
 - *Podpora nástrojov* Nástroje, ktoré možno použiť, siahajú od špeciálnych systémov na správu požiadaviek až po tabuľky a jednoduché databázové systémy.



Riadenie zmeny požiadaviek

- ✧ Rozhodovanie, či by sa mala akceptovať zmena požiadaviek
 - *Analýza problému a špecifikácia zmeny*
 - Počas tejto fázy sa problém alebo návrh zmeny analyzuje, aby sa skontrolovalo, či je platný. Táto analýza sa odošle späť žiadateľovi o zmenu, ktorý môže odpovedať špecifickejším návrhom na zmenu požiadaviek alebo sa môže rozhodnúť žiadosť stiahnuť.
 - *Analýza zmien a kalkulácia*
 - Účinok navrhovanej zmeny sa hodnotí pomocou informácií o sledovateľnosti a všeobecných znalostí systémových požiadaviek. Po dokončení tejto analýzy sa rozhodne, či pokračovať v zmene požiadaviek alebo nie.
 - *Implementácia zmeny*
 - Upraví sa dokument s požiadavkami av prípade potreby aj návrh a implementácia systému. V ideálnom prípade by mal byť dokument usporiadaný tak, aby sa dali jednoducho implementovať zmeny.

Riadenie zmeny požiadaviek





Kapitola 5 – Modelovanie systému

Systémové modelovanie



- ✧ Systémové modelovanie je proces vývoja abstraktných modelov systému, pričom každý model predstavuje iný pohľad na alebo perspektívu daného systému.
- ✧ Systémové modelovanie znamená reprezentovať systém pomocou nejakého druhu grafického zápisu, ktorý je v súčasnosti takmer vždy založený na zápisoch Unified Modeling Language (UML).
- ✧ Modelovanie systému pomáha analytikovi pochopíť funkčnosť systému a modely sa používajú na komunikáciu so zákazníkmi.



Existujúce a plánované modely systémov

- ✧ Pri navrhovaní požiadaviek sa používajú modely existujúceho systému. Pomáhajú objasniť, čo robí existujúci systém, a možno ich použiť ako základ pre diskusiu o jeho silných a slabých stránkach. Tie potom vedú k požiadavkám na nový systém.
- ✧ Modely nového systému sa používajú pri navrhovaní požiadaviek, aby pomohli vysvetliť navrhované požiadavky ostatným účastníkom systému. Inžinieri používajú tieto modely na diskusiu o návrhoch dizajnu a na dokumentáciu systému na implementáciu.
- ✧ V **modelom riadenom inžinierskom (MDE)** procese je možné z modelu systému vygenerovať úplnú alebo čiastočnú implementáciu systému.



- ✧ **Unified** (Booch , Rumbaugh , Jacobson)
Modeling (vizuálny, grafický)
Language (gramatika, syntax, sémantika)

- ✧ Definícia Object Management Group (OMG):
 - Unified Modeling Language je grafický jazyk na **vizualizáciu, špecifikáciu, konštrukciu a dokumentáciu** artefaktov softvérovo náročného systému. UML ponúka štandardný spôsob písania plánov systému vrátane koncepčných vecí, ako sú **obchodné procesy** a **systémové funkcie**, ako aj konkrétnych vecí, ako sú **príkazy programovacieho jazyka**, **schémy databázy** a **opakovane použiteľné softvérové komponenty** .

História UML



- ✧ 1996 : verzia 0.9 – pridali sa IBM, HP, MS, Oracle,...
- ✧ 1997: UML štandardizované spoločnosťou Object Management Group (OMG), verzie 1.0 a 1.1
- ✧ 1998-2001: verzie 1.2, 1.3, 1.4 – malé zmeny
- ✧ ISO/IEC 19501:2005 (UML verzia 1.4.2)
- ✧ 2005: verzia 2.0
- ✧ Aktuálna verzia 2.5.1 od decembra 2017

Systémové perspektívy

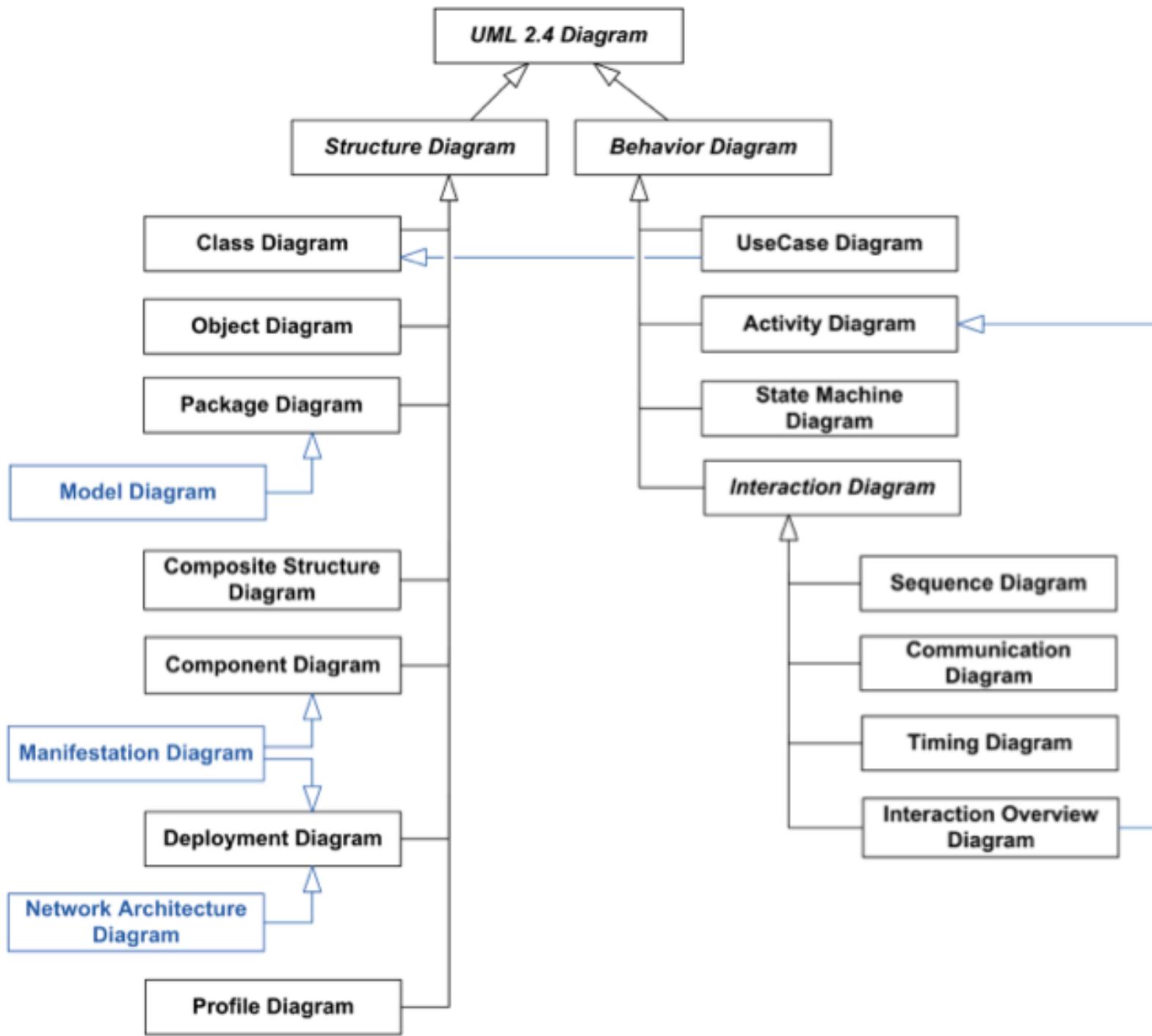


- ✧ Externá perspektíva, kde modelujete **kontext** alebo prostredie systému z vonkajšieho pohľadu.
- ✧ Interakčná perspektíva, kde modelujete interakcie medzi systémom a jeho prostredím alebo medzi komponentmi systému.
- ✧ Štrukturálna perspektíva, kde modelujete organizáciu systému alebo štruktúru údajov, ktoré systém spracováva.
- ✧ Perspektíva správania, kde modelujete dynamické správanie systému a ako reaguje na udalosti.



Typy diagramov UML

- ✧ **Diagramy aktivít (činností)**, ktoré zobrazujú činnosti zahrnuté v procese alebo pri spracovaní údajov.
- ✧ **Diagramy prípadov použitia**, ktoré zobrazujú interakcie medzi systémom a jeho prostredím.
- ✧ **Sekvenčné diagramy**, ktoré zobrazujú interakcie medzi aktérmi a systémom a medzi komponentmi systému.
- ✧ **Diagramy tried**, ktoré zobrazujú triedy objektov v systéme a asociácie medzi týmito triedami.
- ✧ **Stavové diagramy**, ktoré ukazujú, ako systém reaguje na vnútorné a vonkajšie udalosti.



Použitie grafických modelov



- ✧ Ako prostriedok na **uľahčenie diskusie** o existujúcom alebo navrhovanom systéme
 - Neúplné a čiastočne nesprávne modely sú v poriadku, pretože ich úlohou je podporovať diskusiu.
- ✧ Ako spôsob **dokumentácie** existujúceho systému
 - Modely by mali byť presnou reprezentáciou systému, ale nemusia byť úplné.
- ✧ Ako podrobný **popis** systému, ktorý možno použiť na vygenerovanie implementácie systému
 - Modely musia byť správne aj úplné

(1) Externá perspektíva - Kontextové modely



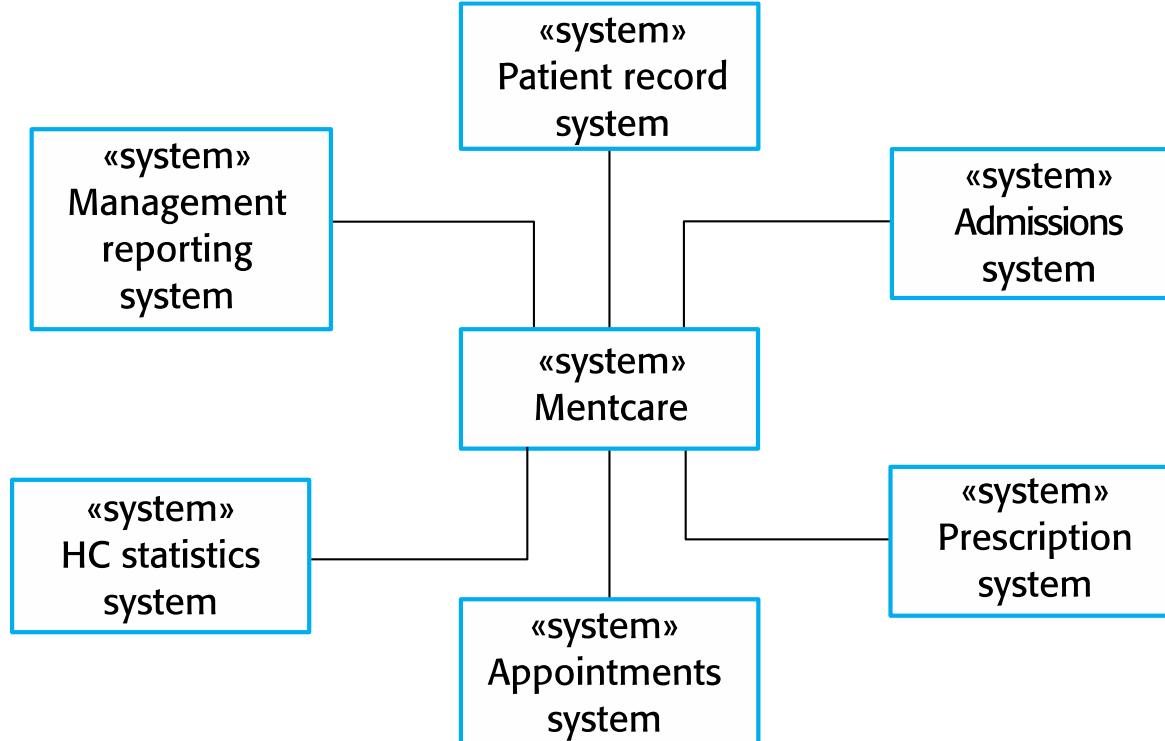
- ✧ **Kontextové modely** sa používajú na ilustráciu operačného kontextu systému – ukazujú, čo leží mimo hraníc systému.
- ✧ Sociálne a organizačné obavy môžu ovplyvniť rozhodnutie o umiestnení hraníc systému.
- ✧ **Architektonické modely** ukazujú systém a jeho vzťah s inými systémami.

Hranice systému



- ✧ Hranice systému sú stanovené tak, aby definovali, čo je vnútri a čo je mimo systém.
 - Zobrazujú ďalšie systémy, ktoré sa používajú alebo závisia od vyvíjaného systému.
- ✧ Poloha systémovej hranice má zásadný vplyv na systémové požiadavky.
- ✧ Definovanie systémovej hranice je politický úsudok
 - Môžu existovať tlaky na vytvorenie systémových hraníc, ktoré zvyšujú/znižujú vplyv alebo pracovné zaťaženie rôznych častí organizácie.

Kontext systému Mentcare



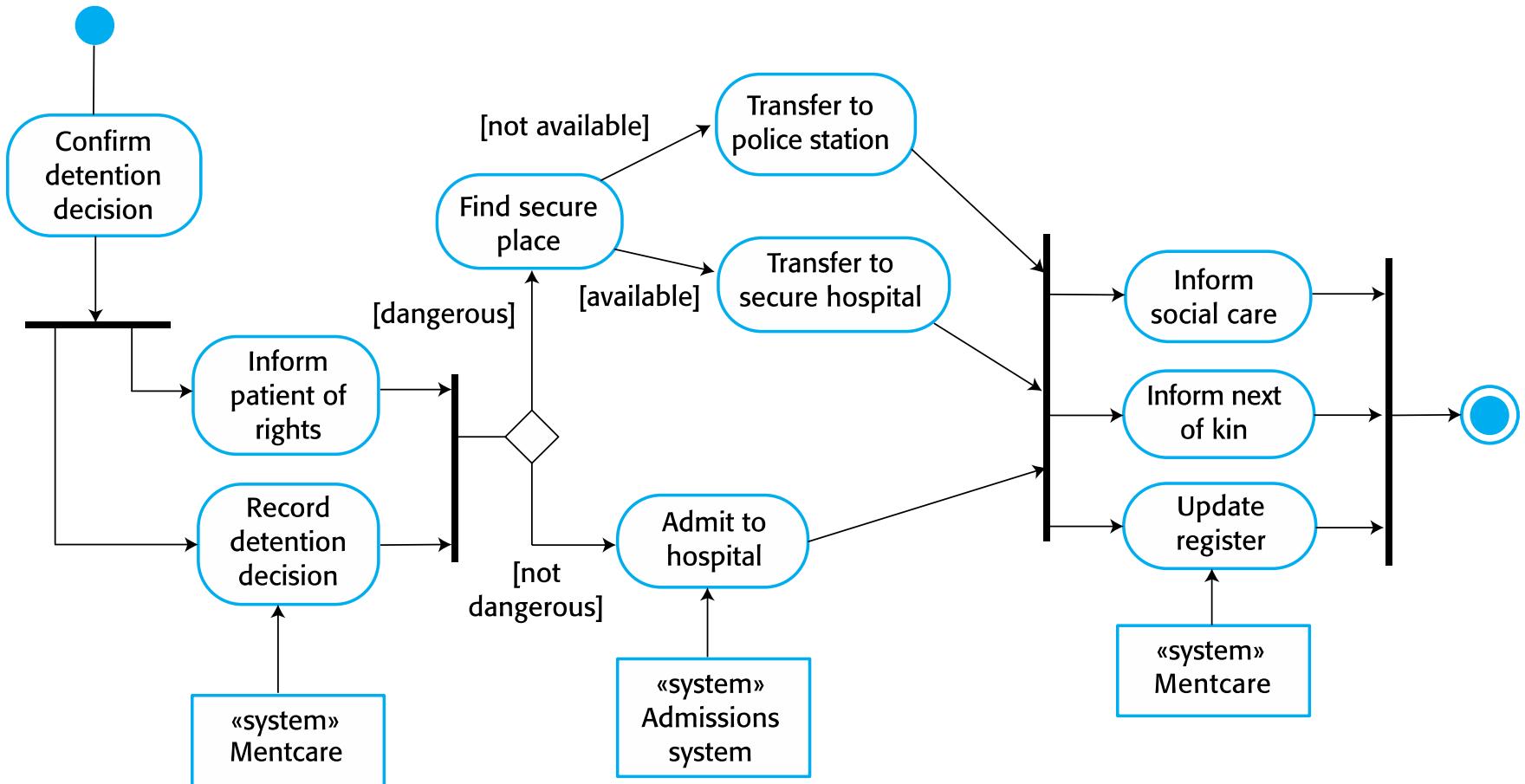
- Box-and-line diagram (škatuľky a čiary)
- UML diagram tried – kde triedy sú napr. <<system>>



(2) Procesná perspektíva

- ✧ Kontextové modely jednoducho ukazujú ostatné systémy v prostredí, nie to, ako sa vyvíjaný systém používa v tomto prostredí.
- ✧ **Procesné modely** odhalujú, ako sa vyvíjaný systém používa v širších obchodných procesoch.
- ✧ Na definovanie sa môžu použiť diagramy aktivít UML
 - Modely podnikových procesov (**workflow**).
 - Modely transformácií údajov (**dataflow**).

Procesný model nedobrovoľného zadržania





(3) Interakčné modely

- ✧ Modelovanie interakcie používateľov je dôležité, pretože pomáha identifikovať požiadavky používateľov.
- ✧ Modelovanie interakcie medzi systémami poukazuje na komunikačné problémy, ktoré môžu vzniknúť.
- ✧ Modelovanie interakcie komponentov nám pomáha pochopiť, či navrhovaná štruktúra systému pravdepodobne poskytne požadovaný výkon a spoľahlivosť systému.
- ✧ Na modelovanie interakcie možno použiť UML diagramy prípadov použitia a diagramy sekvencií a komunikácie .

Modelovanie prípadov použitia



- ✧ Prípady použitia boli pôvodne vyvinuté na podporu vyvolávania požiadaviek a teraz sú začlenené do UML.
- ✧ Každý prípad použitia predstavuje samostatnú úlohu, ktorá zahŕňa externú interakciu so systémom.
- ✧ Aktérmi v prípade použitia môžu byť ľudia alebo iné systémy.
- ✧ Znázorňované diagramom s cieľom poskytnúť prehľad prípadu použitia a v podrobnejšej textovej forme.

Prípad použitia "prenosu údajov"



- ✧ Prípad použitia v Mentcare systéme



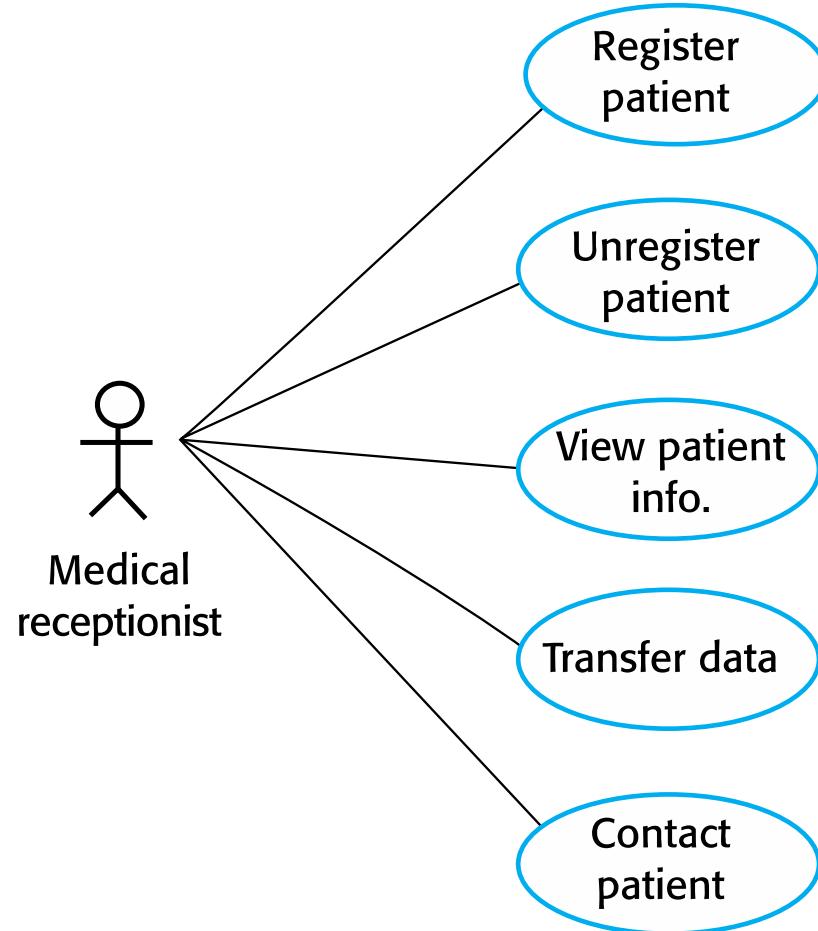
Tabuľkový popis prípadu použitia "prenos údajov".



MHC -PMS: Prenos dát

Herci	Lekárska recepčná, systém záznamov o pacientoch (PRS)
Popis	Recepčný môže preniesť údaje zo systému Mentcase do všeobecnej databázy záznamov o pacientoch, ktorú spravuje zdravotnícky úrad. Prenesené informácie môžu byť buď aktualizované osobné údaje (adresa, telefónne číslo atď.) alebo súhrn pacientovej diagnózy a liečby.
Údaje	Osobné údaje pacienta, súhrn liečby
Stimulácia	Užívateľský príkaz vydaný lekárskou recepciou
odpoveď	Potvrdenie, že PRS bola aktualizovaná
Komentáre	Recepčný musí mať príslušné bezpečnostné povolenia na prístup k informáciám o pacientovi a PRS .

Prípady použitia v systéme Mentcare zahŕňajúce hráča „recepčný“



Sekvenčné diagramy

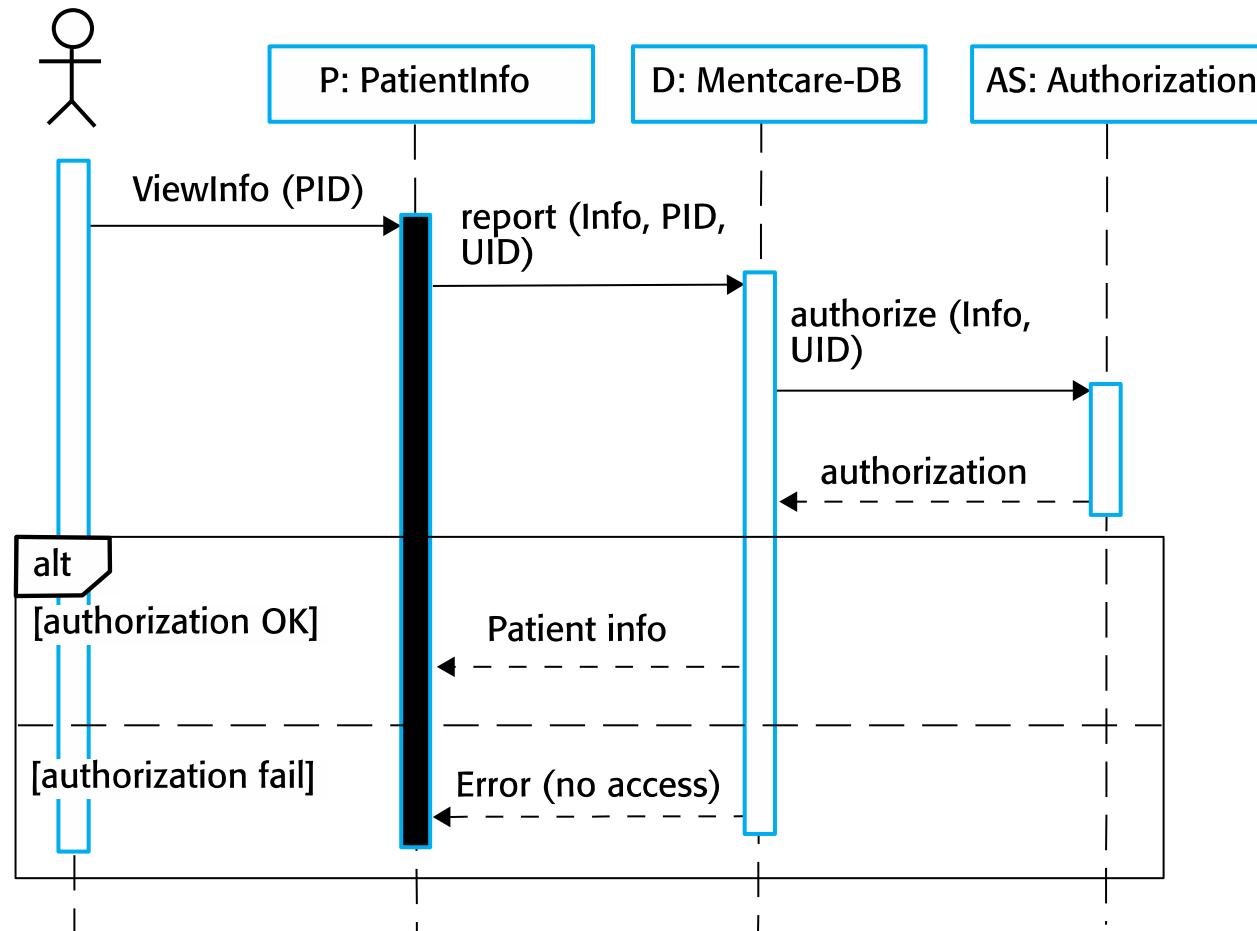


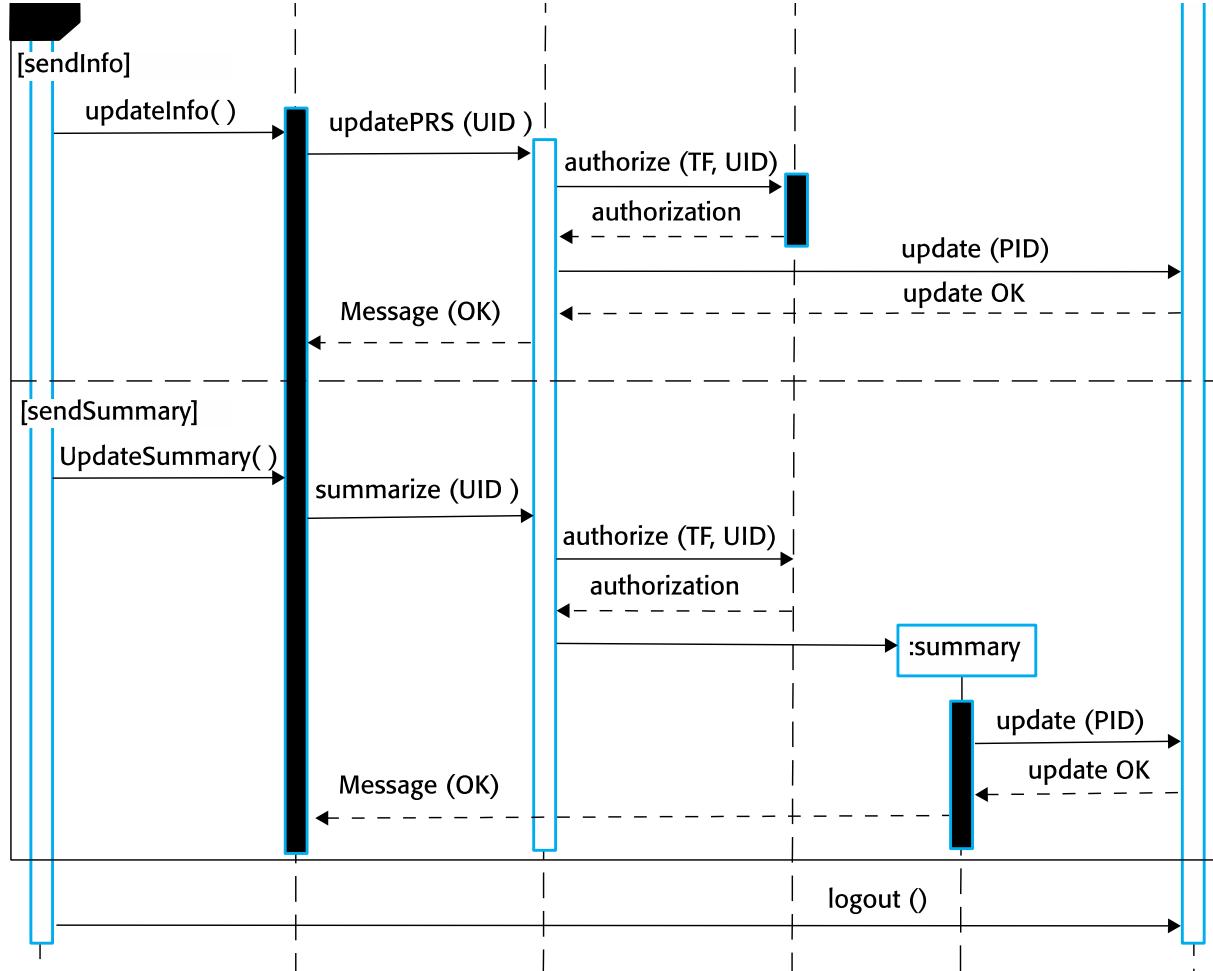
- ✧ Sekvenčné diagramy sú súčasťou UML a používajú sa na modelovanie interakcií medzi aktérmi a objektmi v rámci systému.
- ✧ Sekvenčný diagram zobrazuje postupnosť interakcií, ktoré prebiehajú počas konkrétneho prípadu použitia alebo inštancie prípadu použitia.
- ✧ Predmety a aktéri, ktorých sa to týka, sú uvedené v hornej časti diagramu, pričom z nich je vertikálne nakreslená bodkovaná čiara.
- ✧ Interakcie medzi objektmi sú označené šípkami s poznámkami.

Sekvenčný diagram pre zobrazenie informácií o pacientovi



Medical Receptionist





Sekvenčný
diagram pre
prenos dát

(4) Štrukturálne modely



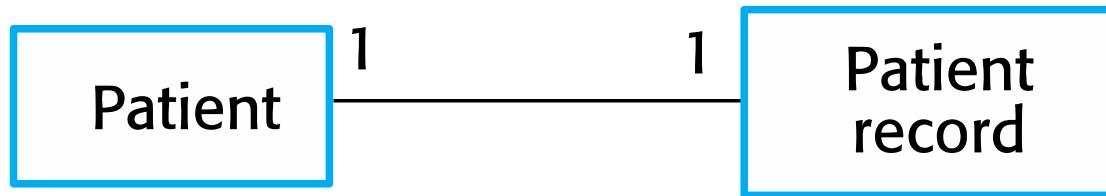
- ✧ Štrukturálne modely softvéru zobrazujú organizáciu systému z hľadiska komponentov, ktoré tvoria tento systém, a ich vzťahov.
- ✧ Štrukturálne modely môžu byť statické modely, ktoré zobrazujú štruktúru návrhu systému, alebo dynamické modely, ktoré zobrazujú organizáciu systému pri jeho vykonávaní.
- ✧ Štrukturálne modely systému vytvárate, keď diskutujete a navrhujete architektúru systému.



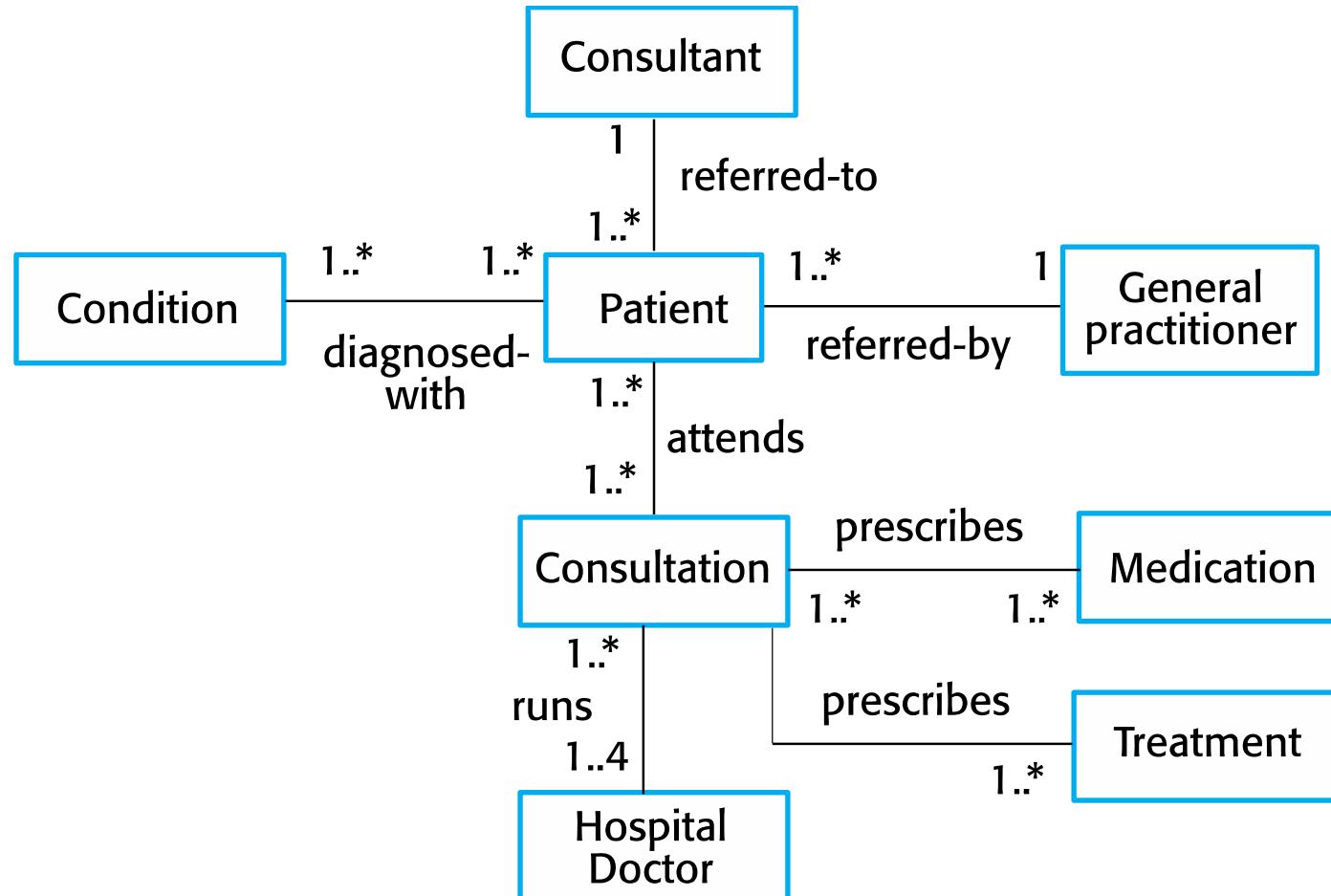
Diagramy tried

- ✧ Diagramy tried sa používajú pri vývoji objektovo orientovaného modelu systému na zobrazenie tried v systéme a asociácií medzi týmito triedami.
- ✧ Triedu objektov si možno predstaviť ako všeobecnú definíciu jedného druhu systémového objektu.
- ✧ Asociácia je prepojenie medzi triedami, ktoré naznačuje, že medzi týmito triedami existuje nejaký vzťah.
- ✧ Keď vyvíjate modely v počiatočných štádiach procesu softvérového inžinierstva, objekty predstavujú niečo v reálnom svete, ako je pacient, predpis, lekár atď.

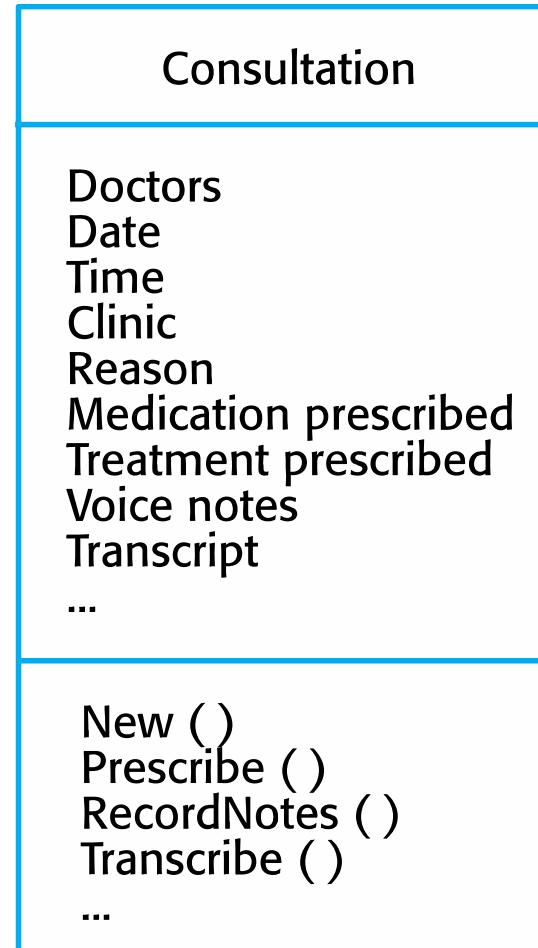
UML triedy a asociácie



Triedy a združenia v Mentcare



Konzultačná trieda



Zovšeobecnenie



- ✧ Zovšeobecnenie je každodenná technika, ktorú používame na zvládnutie zložitosti.
- ✧ Namiesto toho, aby sme sa učili podrobné charakteristiky každej entity, ktorú zažívame, zaraďujeme tieto entity do všeobecnejších tried (zvieratá, autá, domy atď.) a učíme sa charakteristiky týchto tried.
- ✧ To nám umožňuje odvodiť, že rôzni členovia týchto tried majú niektoré spoločné vlastnosti, napr. veveričky a potkany sú hlodavce.

Zovšeobecnenie



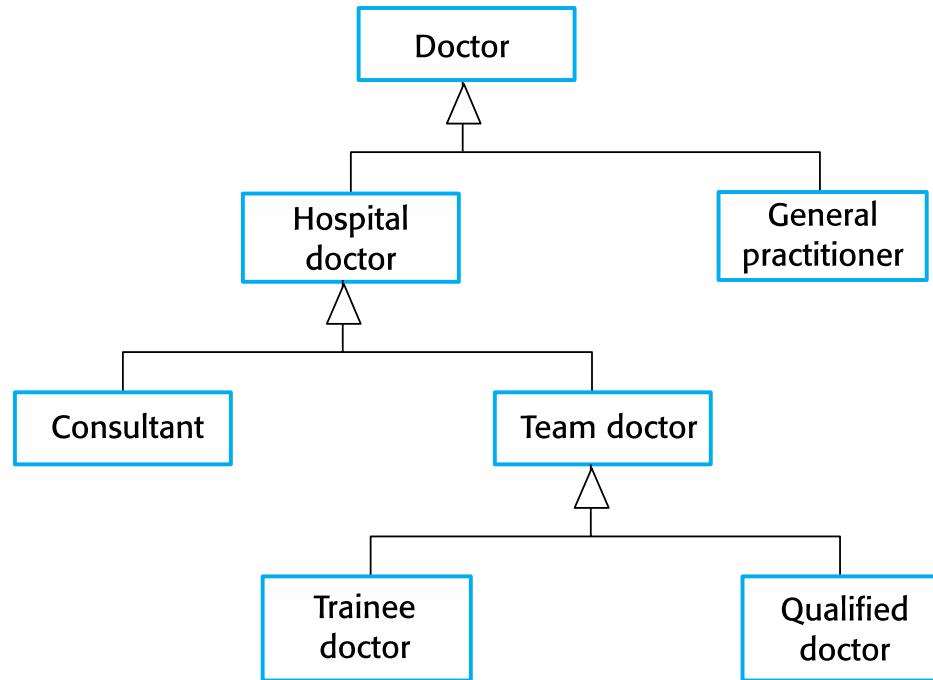
- ✧ Pri modelovaní systémov je často užitočné preskúmať triedy v systéme, aby ste zistili, či existuje priestor na zovšeobecnenie. Ak sú navrhnuté zmeny, nemusíte sa pozerat' na všetky triedy v systéme, aby ste zistili, či sa ich zmena týka.
- ✧ V objektovo orientovaných jazykoch, ako je Java, sa zovšeobecnenie implementuje pomocou mechanizmov dedičnosti tried zabudovaných do jazyka.
- ✧ Vo všeobecnosti sú atribúty a operácie spojené s triedami vyšej úrovne tiež spojené s triedami nižšej úrovne.
- ✧ Triedy nižšej úrovne sú podtriedy, ktoré zdedia atribúty a operácie zo svojich nadtried . Tieto triedy nižšej úrovne potom pridávajú špecifickejšie atribúty a operácie.



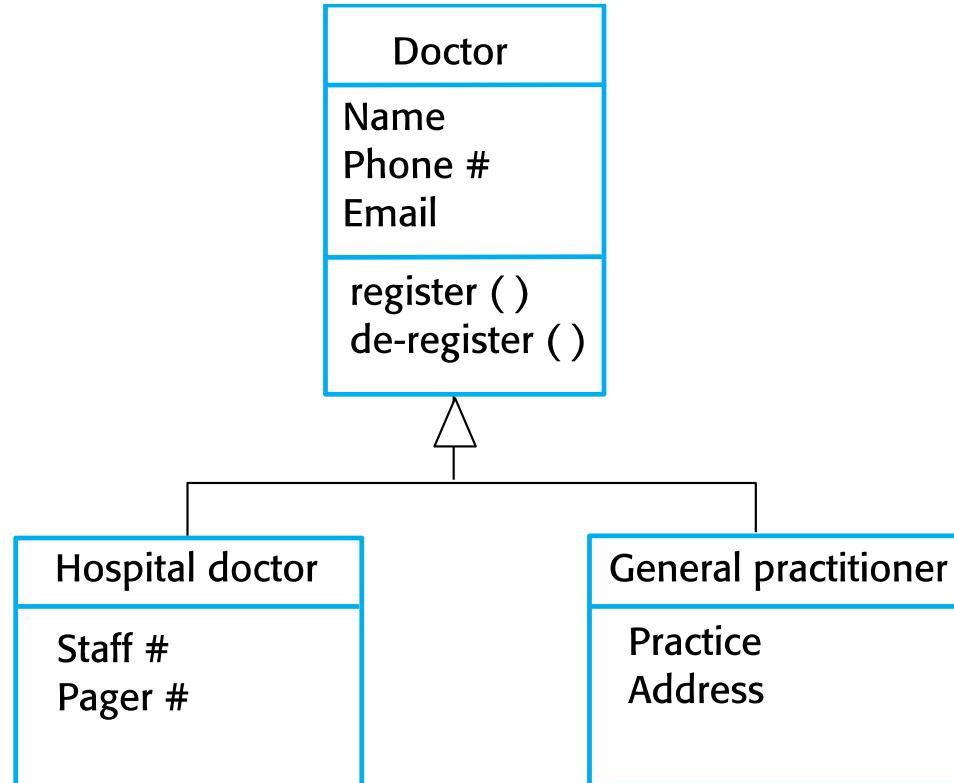
Hierarchia zovšeobecnenia (opak dedenia)

Software Engineering
6th edition

Ian Sommerville



Hierarchia zovšeobecnenia s pridanými detailmi

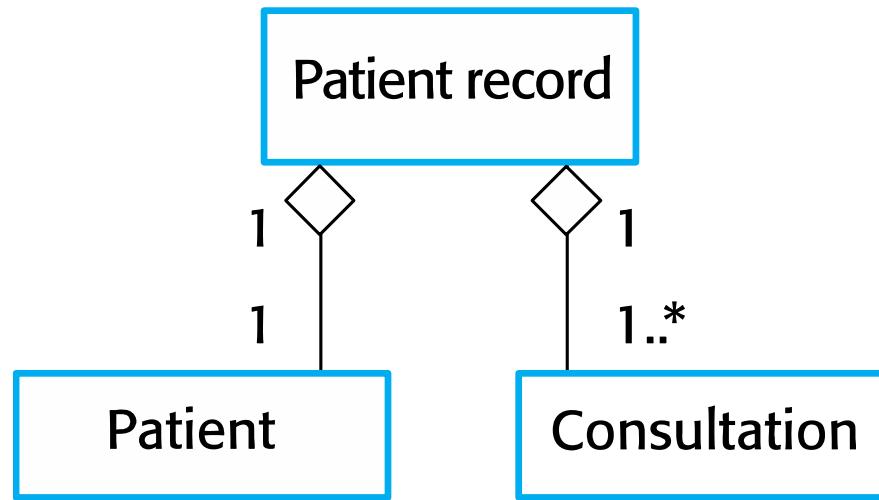


Modely agregácie tried



- ✧ Agregačný model ukazuje, ako sa triedy, ktoré sú kolekciami, skladajú z iných tried.
- ✧ Agregačné modely sú podobné ako časť vztahu v modeloch sémantických údajov.

Agregačná asociácia



(5) Modely správania



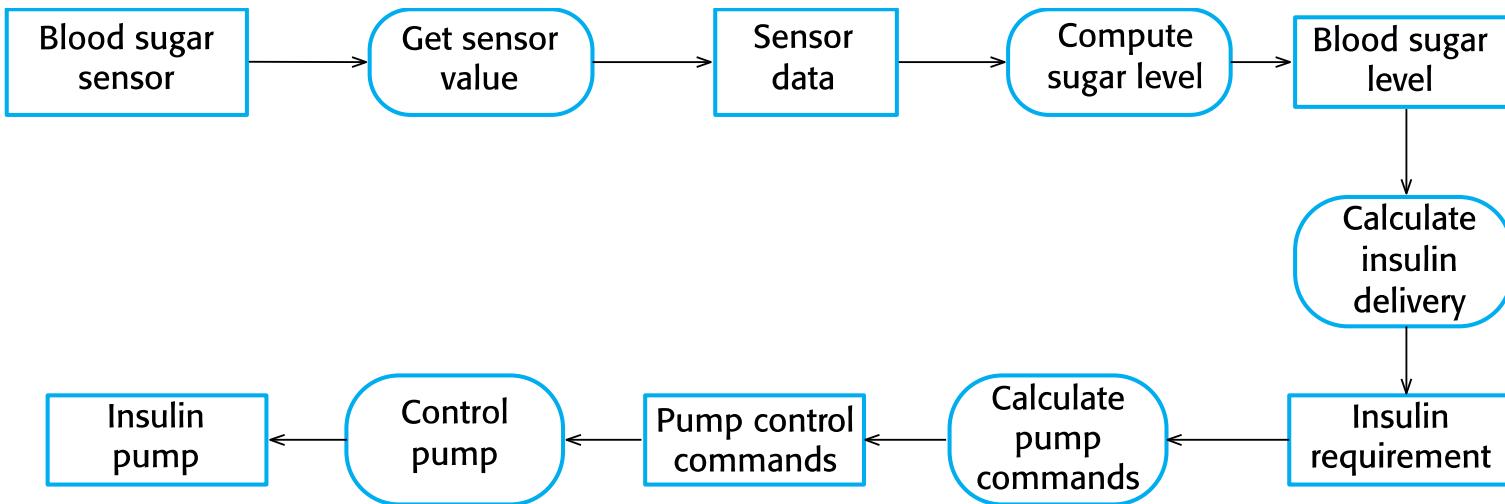
- ✧ Behaviorálne modely (chovania sa, správania) sú modely dynamického správania sa systému pri jeho vykonávaní. Ukazujú, čo sa stane alebo čo sa má stať, keď systém zareaguje na podnet zo svojho okolia.
- ✧ Tieto stimuly si môžete predstaviť ako dva typy:
 - **Údaje:** Prichádzajú nejaké údaje, ktoré musí systém spracovať.
 - **Udalosti:** Nastane nejaká udalosť, ktorá spustí spracovanie systémom. Udalosti môžu mať súvisiace údaje, aj keď to nie je vždy tak.

Modelovanie založené na údajoch



- ✧ Mnohé obchodné systémy sú systémy na spracovanie údajov, ktoré sú primárne poháňané údajmi. Sú riadené vstupom údajov do systému s relatívne malým externým spracovaním udalostí.
- ✧ Modely založené na údajoch ukazujú postupnosť akcií zapojených do spracovania vstupných údajov a generovania súvisiaceho výstupu - **dataflow**.
- ✧ Sú obzvlášť užitočné pri analýze požiadaviek, pretože sa dajú použiť na zobrazenie komplexného spracovania v systéme.

Model činnosti činnosti inzulínovej pumpy

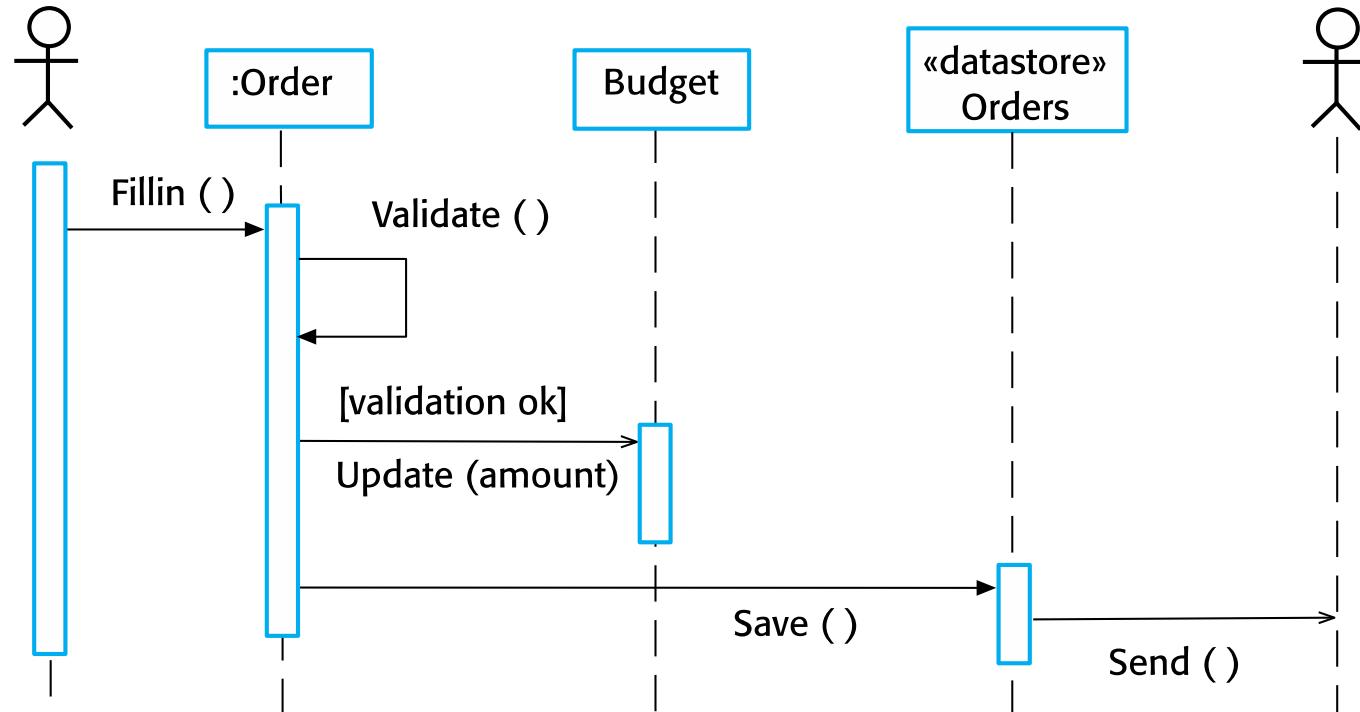


Spracovanie objednávky



Purchase officer

Supplier



Modelovanie riadené udalosťami



- ✧ Systémy v reálnom čase sú často riadené udalosťami s minimálnym spracovaním údajov. Napríklad systém prepínania pevnej linky reaguje na udalosti, ako je „zdvihnutie sluchátka“ generovaním oznamovacieho tónu.
- ✧ **Modelovanie riadené udalosťami** ukazuje, ako systém reaguje na vonkajšie a vnútorné udalosti.
- ✧ Vychádza z predpokladu, že systém má konečný počet stavov a že udalosti (podnety) môžu spôsobiť prechod z jedného stavu do druhého.

Modely stavových strojov



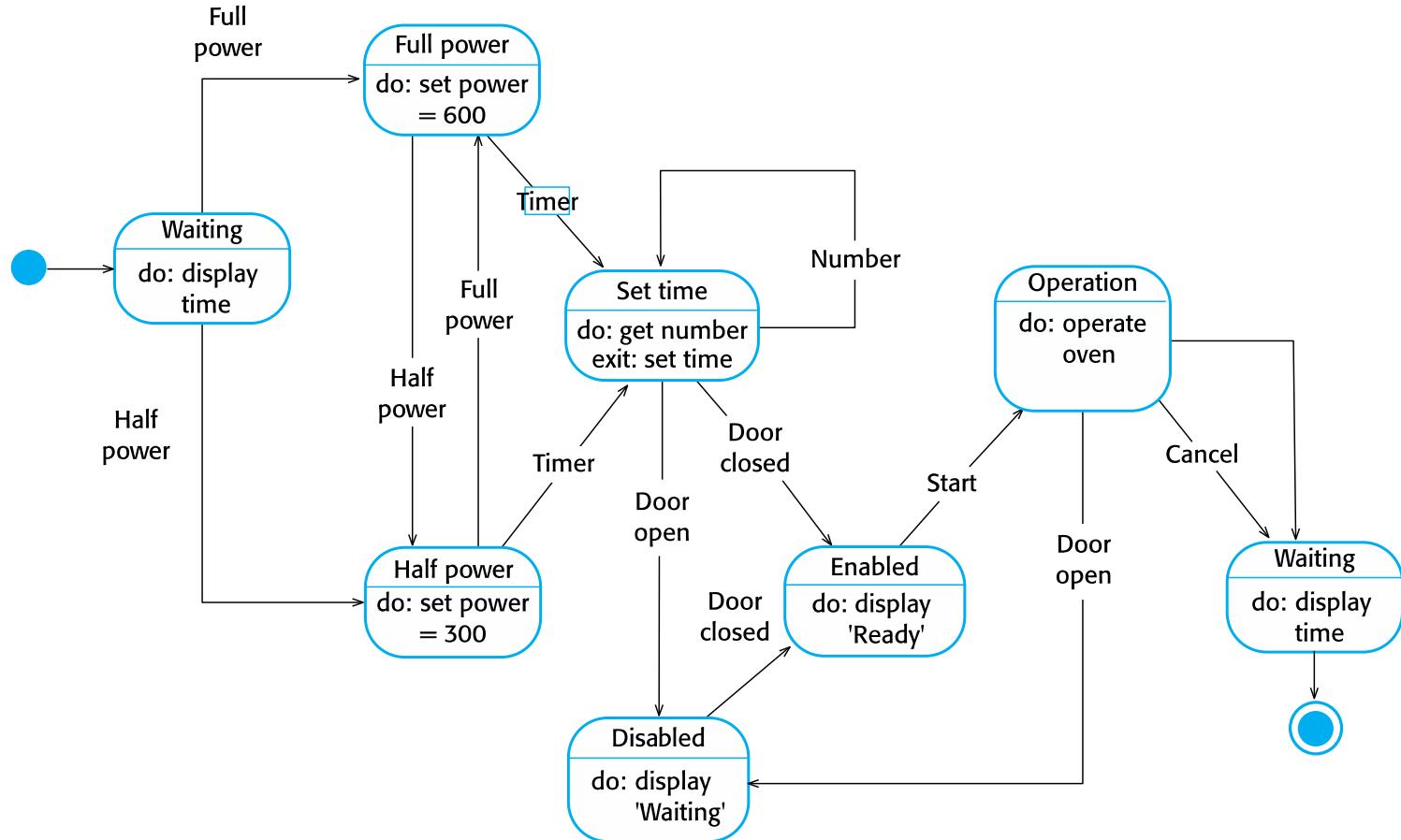
- ✧ Tieto modelujú správanie systému v reakcii na vonkajšie a vnútorné udalosti.
- ✧ Zobrazujú reakcie systému na podnety, takže sa často používajú na modelovanie systémov v reálnom čase.
- ✧ Modely stavových strojov zobrazujú stavy systému ako uzly a udalosti ako oblúky medzi týmito uzlami. Ked' dôjde k udalosti, systém sa presunie z jedného stavu do druhého.
- ✧ Stavové diagramy sú neoddeliteľnou súčasťou UML a používajú sa na reprezentáciu modelov stavových strojov.

Stavový diagram mikrovlnnej rúry



Software Engineering
6th edition

Ian Sommerville

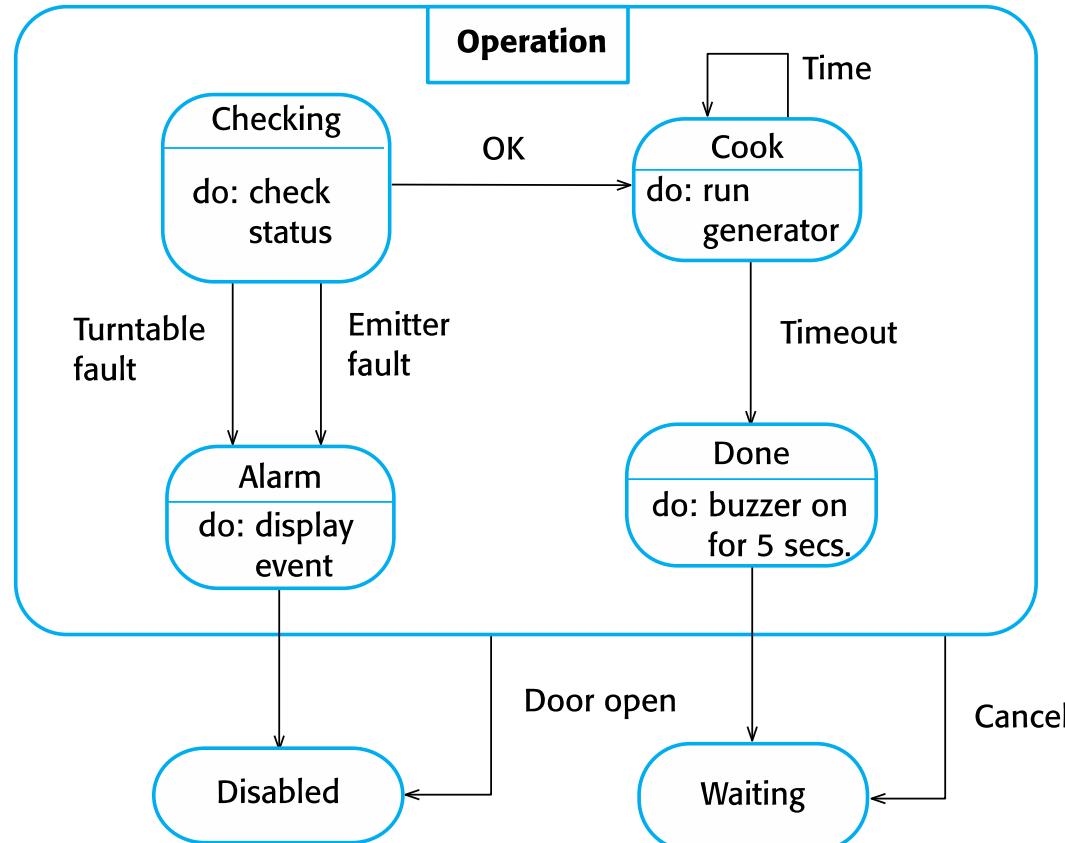


Stav "Prevádzka" mikrovlnnej rúry



Software Engineering
6th edition

Ian Sommerville



Stavy pre mikrovlnnú rúru



Štát	Popis
Čakanie	Rúra čaká na vstup. Na displeji sa zobrazí aktuálny čas.
Polovičný výkon	Výkon rúry je nastavený na 300 wattov. Na displeji sa zobrazí „Polovičný výkon“.
Plný výkon	Výkon rúry je nastavený na 600 wattov. Na displeji sa zobrazí „Plný výkon“.
Nastav čas	Čas varenia je nastavený na hodnotu zadanú používateľom. Displej zobrazuje zvolený čas varenia a aktualizuje sa podľa nastavenia času.
Zakázané	Prevádzka rúry je z bezpečnostných dôvodov vypnutá. Vnútorné osvetlenie rúry je zapnuté. Displej zobrazuje 'Nie je pripravený'.
Povolené	Prevádzka rúry je aktivovaná. Vnútorné osvetlenie rúry je vypnuté. Displej zobrazuje „Pripravené na varenie“.
Prevádzka	Rúra v prevádzke. Vnútorné osvetlenie rúry je zapnuté. Displej zobrazuje odpočítavanie časovača. Po dokončení varenia zaznie na päť sekúnd bzučiak. Osvetlenie rúry je zapnuté. Displej zobrazuje 'Varenie je dokončené', zatiaľ čo znie bzučiak .
30.10.2014	2

Prechody pre mikrovlnnú rúru



Software Engineering
6th edition

Ian Sommerville

Stimulácia	Popis
Polovičný výkon	Používateľ stlačil tlačidlo polovičného napájania.
Plný výkon	Používateľ stlačil tlačidlo plného napájania.
Časovač	Používateľ stlačil jedno z tlačidiel časovača.
číslo	Používateľ stlačil číselnú klávesu.
Dvere otvorené	Spínač dvierok rúry nie je zatvorený.
Dvere zatvorené	Spínač dvierok rúry je zatvorený.
Štart	Používateľ stlačil tlačidlo Štart.
Zrušiť'	Používateľ stlačil tlačidlo Zrušiť' .

Modelom riadené inžinierstvo



- ✧ Modelom riadené inžinierstvo (MDE) je prístup k vývoju softvéru, kde hlavnými výstupmi vývojového procesu sú skôr modely ako programy.
- ✧ Programy, ktoré sa spúšťajú na hardvérovej/softvérovej platforme, sa potom generujú automaticky z modelov.
- ✧ Zástancovia MDE tvrdia, že to zvyšuje úroveň abstrakcie v softvérovom inžinierstve, takže inžinieri sa už nemusia zaoberať detailmi programovacieho jazyka alebo špecifikami vykonávacích platforiem.



Použitie modelom riadeného inžinierstva

✧ Modelom riadené inžinierstvo je stále v ranom štádiu vývoja a nie je jasné, či bude mať významný vplyv na prax softvérového inžinierstva alebo nie.

✧ Výhody

- Umožňuje posúdiť systémy na vyšších úrovniah abstrakcie
- Automatické generovanie kódu znamená, že je lacnejšie prispôsobiť systémy novým platformám.

✧ Nevýhody

- Modely pre abstrakciu nie sú úplne vhodné na implementáciu.
- Úspory z generovania kódu môžu byť vyvážené nákladmi na vývoj prekladačov pre nové platformy.

Architektúra riadená modelom



- ✧ Modelom riadená architektúra (MDA) bola predchodom všeobecnejšieho modelom riadeného inžinierstva
- ✧ MDA je modelovo zameraný prístup k návrhu a implementácii softvéru, ktorý využíva podmnožinu modelov UML na popis systému.
- ✧ Vytvárajú sa modely na rôznych úrovniach abstrakcie. Z vysokoúrovňového, platformovo nezávislého modelu je v princípe možné vygenerovať pracovný program bez manuálneho zásahu.



Typy modelu

✧ Model nezávislý na výpočte (CIM)

- Tieto modelujú dôležité doménové abstrakcie používané v systéme. CIM sa niekedy nazývajú doménové modely.

✧ Model nezávislý na platforme (PIM)

- Tieto modelujú fungovanie systému bez odkazu na jeho implementáciu. PIM je zvyčajne opísaný pomocou modelov UML, ktoré ukazujú statickú štruktúru systému a ako reaguje na vonkajšie a vnútorné udalosti.

✧ Modely špecifické pre platformu (PSM)

- Ide o transformácie modelu nezávislého na platforme so samostatným PSM pre každú aplikačnú platformu. V zásade môžu existovať vrstvy PSM, pričom každá vrstva pridáva určité detaily špecifické pre platformu.

Agilné metódy a MDA



- ✧ Vývojári MDA tvrdia, že je určený na podporu iteratívneho prístupu k vývoju, a preto ho možno použiť v rámci agilných metód.
- ✧ Pojem rozsiahleho dopredu vykonaného modelovania je v rozpore so základnými myšlienkami v agilnom manifeste a je podozrenie, že len málo agilných vývojárov sa stotožňuje s modelom riadeným inžinierstvom.

Prijatie MDA



- ✧ Prijatie MDE/MDA obmedzuje celý rad faktorov
- ✧ Na konverziu modelov z jednej úrovne na druhú je potrebná podpora špeciálnych nástrojov
- ✧ Dostupnosť nástrojov je obmedzená a organizácie môžu vyžadovať prispôsobenie nástrojov a prispôsobenie ich prostrediu
- ✧ Pokial' ide o systémy s dlhou životnosťou vyvinuté pomocou MDA, spoločnosti sa zdráhajú vyvíjať svoje vlastné nástroje alebo sa spoliehať na malé spoločnosti, ktoré môžu skončiť s podnikaním.

Prijatie MDA



- ✧ Modely sú dobrým spôsobom, ako uľahčiť diskusie o návrhu softvéru. Abstrakcie, ktoré sú užitočné pre diskusie, však nemusia byť tými správnymi abstrakciami na implementáciu.
- ✧ V prípade najkomplexnejších systémov nie je implementácia hlavným problémom – dôležitejšie sú inžinierstvo požiadaviek, bezpečnosť a spoľahlivosť, integrácia so staršími systémami a testovanie.

Prijatie MDA



- ✧ Argumenty pre platformovú nezávislosť platia len pre veľké systémy s dlhou životnosťou. V prípade softvérových produktov a informačných systémov budú úspory vyplývajúce z používania MDA pravdepodobne prevážené nákladmi na jeho zavedenie a nástroje.
- ✧ Široké prijímanie agilných metód v rovnakom období, v ktorom sa MDA vyvíjalo, odviedlo pozornosť od prístupov založených na modeloch.
- ✧ V praxi sa teda najčastejšie stretnete s týmito (v poradí):
 - Model-driven analysis
 - Model-driven architecture
 - Model-driven engineering



Kapitola 6 – Návrh architektúry

Architektonický dizajn



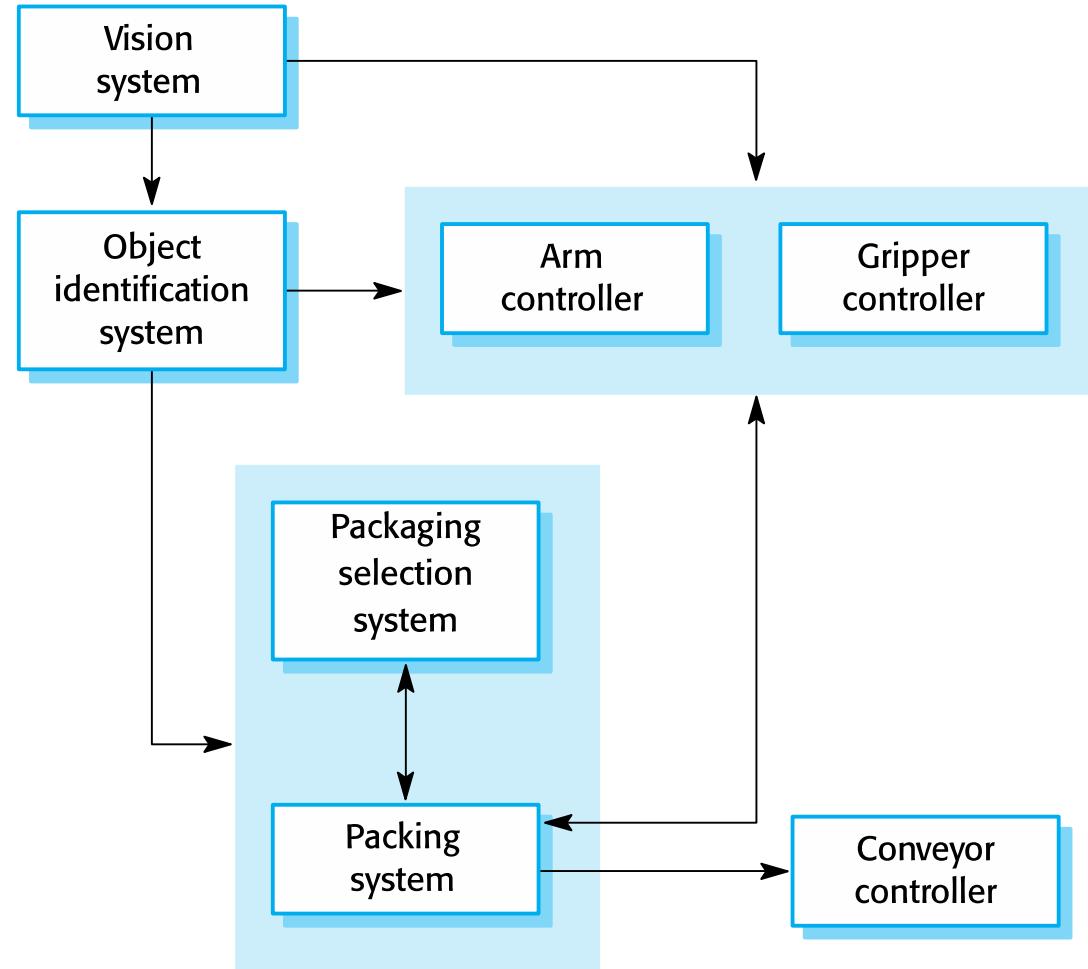
- ✧ Architektonický dizajn (návrh) sa zaoberá pochopením toho, ako by mal byť softvérový systém organizovaný, a navrhnutím celkovej štruktúry tohto systému.
- ✧ Architektonický dizajn je kritickým spojením medzi návrhom a inžinierstvom požiadaviek, pretože identifikuje hlavné konštrukčné komponenty v systéme a vzťahy medzi nimi.
- ✧ Výstupom procesu architektonického návrhu je **model architektúry**, ktorý popisuje, ako je systém organizovaný ako súbor komunikujúcich komponentov.

Agile a architektúra



- ✧ Všeobecne platí, že počiatočným štádiom agilných procesov je návrh celkovej architektúry systému.
- ✧ Refaktorovanie architektúry systému je zvyčajne drahé, pretože ovplyvňuje veľa komponentov v systéme.

Architektúra riadiaceho systému baliaceho robota





Architektonická abstrakcia

- ✧ Architektúra v malom sa zaoberá architektúrou jednotlivých programov. Na tejto úrovni sa zaoberáme spôsobom, akým je individuálny program rozložený na komponenty.
- ✧ Architektúra vo veľkom sa zaoberá architektúrou komplexných podnikových systémov, ktoré zahŕňajú iné systémy, programy a programové komponenty. Tieto podnikové systémy sú distribuované na rôznych počítačoch, ktoré môžu vlastniť a spravovať rôzne spoločnosti.



Výhody explicitnej architektúry

✧ Komunikácia so zainteresovanými stranami

- Architektúra môže byť použitá pre diskusiu medzi zainteresovanými stranami systému.

✧ Systémová analýza

- Znamená to, že je možná analýza, či systém dokáže splniť svoje nefunkcionálne požiadavky.

✧ Opäťovné použitie vo veľkom meradle

- Architektúra môže byť opakovane použiteľná v celom rade systémov
- Môžu byť vyvinuté architektúry produktového radu.



Použitie architektonických modelov

- ✧ Ako spôsob **uľahčenia diskusie o návrhu systému**
 - Architektonický pohľad na systém na vysokej úrovni je užitočný pre komunikáciu so zainteresovanými stranami systému a plánovanie projektu, pretože nie je preplnený detailmi. Zainteresované strany sa s tým môžu stotožniť a pochopiť abstraktný pohľad na systém. Potom môžu diskutovať o systéme ako celku bez toho, aby boli zmätení detailmi.
- ✧ Ako spôsob **dokumentovania architektúry**, ktorá bola navrhnutá
 - Cieľom je vytvoriť kompletný model systému, ktorý zobrazuje rôzne komponenty v systéme, ich rozhrania a prepojenia.



Architektonické reprezentácie

- ✧ Krabicové a čiarové diagramy sú veľmi abstraktné - nezobrazujú povahu vzťahov komponentov ani externe viditeľné vlastnosti podsystémov.
 - Užitočné však pre komunikáciu so zainteresovanými stranami a pre plánovanie projektov.
- ✧ Niektorí ľudia tvrdia, že Unified Modeling Language (UML) je vhodnou notáciou na popis a dokumentáciu systémových architektúr.
- ✧ Architektonické popisné jazyky (ADL) boli vyvinuté, ale nie sú široko používané

Rozhodnutia o architektonickom návrhu



- ✧ Architektonický návrh je kreatívny proces, takže proces sa líši v závislosti od typu vyvíjaného systému.
- ✧ Avšak množstvo spoločných rozhodnutí pokrýva všetky procesy návrhu a tieto rozhodnutia ovplyvňujú nefunkcionálne charakteristiky systému.

Rozhodnutia o architektonickom dizajne



Is there a generic application architecture that can act as a template for the system that is being designed?

How will the system be distributed across hardware cores or processors?

What architectural patterns or styles might be used?

What will be the fundamental approach used to structure the system?

What strategy will be used to control the operation of the components in the system?

How will the structural components in the system be decomposed into sub-components?

What architectural organization is best for delivering the non-functional requirements of the system?

How should the architecture of the system be documented?

Opäťovné použitie architektúry



- ✧ Systémy v rovnakej doméne majú často podobné architektúry, ktoré odrážajú koncepty domény.
- ✧ **Aplikačné produktové rady** sú postavené na základnej architektúre s variantmi, ktoré uspokoja konkrétné požiadavky zákazníkov.
- ✧ Architektúra systému môže byť navrhnutá podľa jedného alebo viacerých architektonických vzorov alebo „štýlov“.
 - Tie zachytávajú podstatu architektúry a môžu byť vytvorené rôznymi spôsobmi.

Charakteristiky architektúry a systému



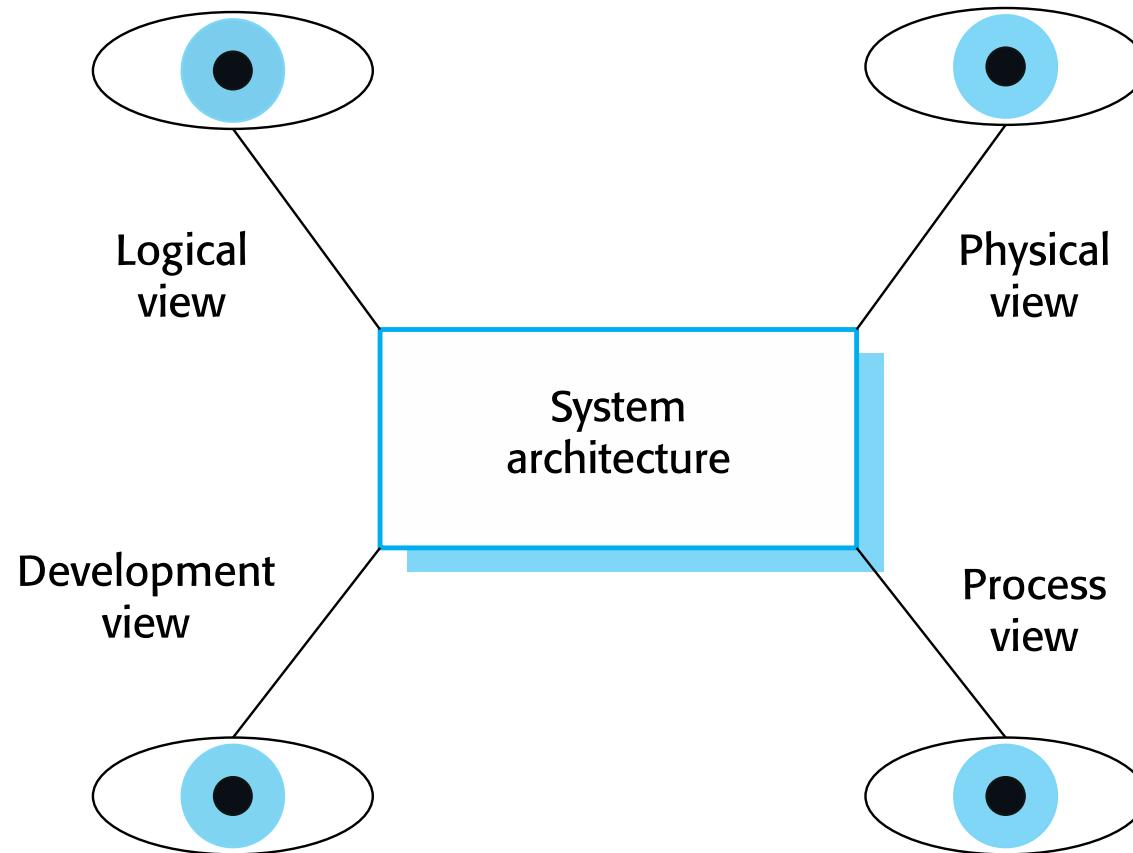
- ✧ Výkon
 - Lokalizovať kritické operácie a minimalizovať komunikáciu. Používajte skôr veľké ako malé komponenty.
- ✧ Bezpečnosť
 - Použite vrstvenú architektúru s kritickými aktívami vo vnútorných vrstvách.
- ✧ Ochrana
 - Lokalizácia prvkov dôležitých pre ochranu v malom počte podsystémov.
- ✧ Dostupnosť
 - Zahrňte redundantné komponenty a mechanizmy na odolnosť voči chybám.
- ✧ Udržiavateľnosť
 - Používajte malé vymeniteľné komponenty.



Architektonické pohľady

- ✧ Aké pohľady alebo perspektívy sú užitočné pri navrhovaní a dokumentovaní architektúry systému?
- ✧ Aké označenia by sa mali použiť na opis architektonických modelov?
- ✧ Každý architektonický model zobrazuje iba jeden pohľad alebo perspektívu systému.
 - Môže to ukázať, ako je systém rozložený na moduly, ako sa vzájomne ovplyvňujú run-time procesy alebo rôzne spôsoby, akými sú systémové komponenty distribuované v sieti. Pre návrh aj dokumentáciu zvyčajne potrebujete prezentovať viacero pohľadov na architektúru softvéru.

Architektonické pohľady



4 + 1 pohľadový model softvérovej architektúry



- ✧ **Logický pohľad**, ktorý zobrazuje kľúčové abstrakcie v systéme ako objekty alebo triedy objektov.
- ✧ **Procesný pohľad**, ktorý ukazuje, ako sa systém za behu skladá z interagujúcich procesov.
- ✧ **Vývojový pohľad**, ktorý ukazuje, ako je softvér rozložený počas vývoja.
- ✧ **Fyzický pohľad**, ktorý zobrazuje hardvér systému a spôsob, akým sú softvérové komponenty distribuované medzi procesormi v systéme.
- ✧ **Súvisiace prípady použitia alebo scenáre (+1)**

Architektonické vzory



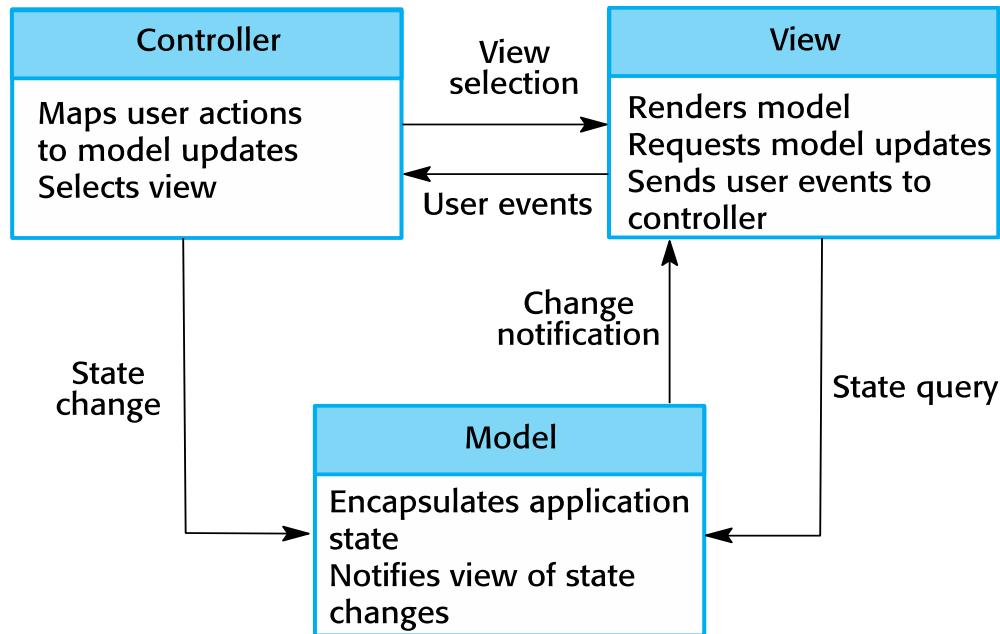
- ✧ Vzory sú prostriedkom na reprezentáciu, zdieľanie a opäťovné použitie vedomostí.
- ✧ **Architektonický vzor** je štylizovaný popis dobrej dizajnérskej praxe, ktorý bol vyskúšaný a testovaný v rôznych prostrediach.
- ✧ Vzory by mali obsahovať informácie o tom, kedy sú a kedy nie sú užitočné.
- ✧ Vzory môžu byť znázornené pomocou tabuľkových a grafických popisov.

Vzor Model-View-Controller (MVC)



názov	MVC (Model-View-Controller)
Popis	Oddeľuje prezentáciu a interakciu od systémových údajov. Systém je štruktúrovaný do troch logických komponentov, ktoré sa navzájom ovplyvňujú. Komponent Model spravuje systémové údaje a súvisiace operácie s týmito údajmi. Komponent View definuje a riadi spôsob, akým sú údaje prezentované používateľovi. Komponent Controller riadi interakciu používateľa (napr. stlačenie klávesov, kliknutie myšou atď.) a tieto interakcie odovzdá do zobrazenia a modelu. Pozri obrázok 6.3.
Príklad	Pozrite si knihu pre webový aplikačný systém organizovaný pomocou vzoru MVC.
Pri použití	Používa sa, keď existuje viacero spôsobov zobrazenia údajov a interakcie s nimi. Používa sa aj vtedy, keď nie sú známe budúce požiadavky na interakciu a prezentáciu údajov.
Výhody	Umožňuje dátam meniť sa nezávisle od ich reprezentácie a naopak. Podporuje prezentáciu rovnakých údajov rôznymi spôsobmi so zmenami vykonanými v jednej reprezentácii zobrazenej vo všetkých.
Nevýhody	Môže zahŕňať dodatočný kód a zložitosť kódu, keď sú dátový model a interakcie jednoduché .

Organizácia Model-View-Controller



(1) Organizácia systému



- ✧ Odráža základnú stratégiu, ktorá sa používa na štruktúrovanie systému.
- ✧ Najčastejšie sa používajú tri organizačné štýly:
 - Štýl **zdieľaného úložiska údajov** ;
 - Štýl **zdieľaných služieb a serverov** ;
 - Abstraktný **strojový alebo vrstvený** štýl.

(1.1) Vrstvená architektúra



- ✧ Používa sa na modelovanie rozhrania podsystémov.
- ✧ Organizuje systém do množiny vrstiev (alebo abstraktných strojov), z ktorých každá poskytuje množinu služieb.
- ✧ Podporuje postupný vývoj podsystémov v rôznych vrstvách. Ked' sa zmení rozhranie vrstvy, ovplyvní to iba susednú vrstvu.
- ✧ Často je to však umelé, aby sa systémy štruktúrovali týmto spôsobom.

Vzor vrstvenej architektúry



názov	Vrstvená architektúra
Popis	Organizuje systém do vrstiev so súvisiacimi funkciami spojenými s každou vrstvou. Vrstva poskytuje služby vrstve nad ňou, takže vrstvy najnižšej úrovne predstavujú základné služby, ktoré sa pravdepodobne budú používať v celom systéme.
Príklad	Vrstvený model systému na zdieľanie autorských dokumentov uchovávaných v rôznych knižniciach.
Pri použití	Používa sa pri budovaní nových zariadení nad existujúcimi systémami; keď je vývoj rozdelený medzi niekoľko tímov, pričom každý tím je zodpovedný za vrstvu funkčnosti; keď existuje požiadavka na viacúrovňové zabezpečenie.
Výhody	Umožňuje výmenu celých vrstiev, pokiaľ je zachované rozhranie. V každej vrstve môžu byť poskytnuté redundantné zariadenia (napr. autentifikácia), aby sa zvýšila spoľahlivosť systému.
Nevýhody	V praxi je zabezpečenie čistého oddelenia medzi vrstvami často zložité a vrstva vysokej úrovne môže musieť interagovať priamo s vrstvami nižšej úrovne než cez vrstvu bezprostredne pod ňou. Výkon môže byť problémom z dôvodu viacerých úrovní interpretácie požiadavky na službu, ktorá sa spracováva na každej vrstve .

Všeobecná vrstvená architektúra



User interface

User interface management
Authentication and authorization

Core business logic/application functionality
System utilities

System support (OS, database etc.)

(1.2) Zdieľané úložisko údajov



- ✧ Subsystémy si musia vymieňať údaje. To možno vykonať dvoma spôsobmi:
 - **(1.2) Centrálne úložisko:** Zdieľané údaje sa uchovávajú v centrálnej databáze alebo úložisku a môžu k nim pristupovať všetky podsystémy;
 - **(1.3) Distribuované úložiská:** Každý podsystém udržiava svoju vlastnú databázu a údaje explicitne odovzdáva iným podsystémom.
- ✧ Ked' sa majú zdieľať veľké množstvá údajov, najčastejšie sa používa model zdieľania úložiska, čo je účinný mechanizmus zdieľania údajov.

Vzor centrálneho úložiska



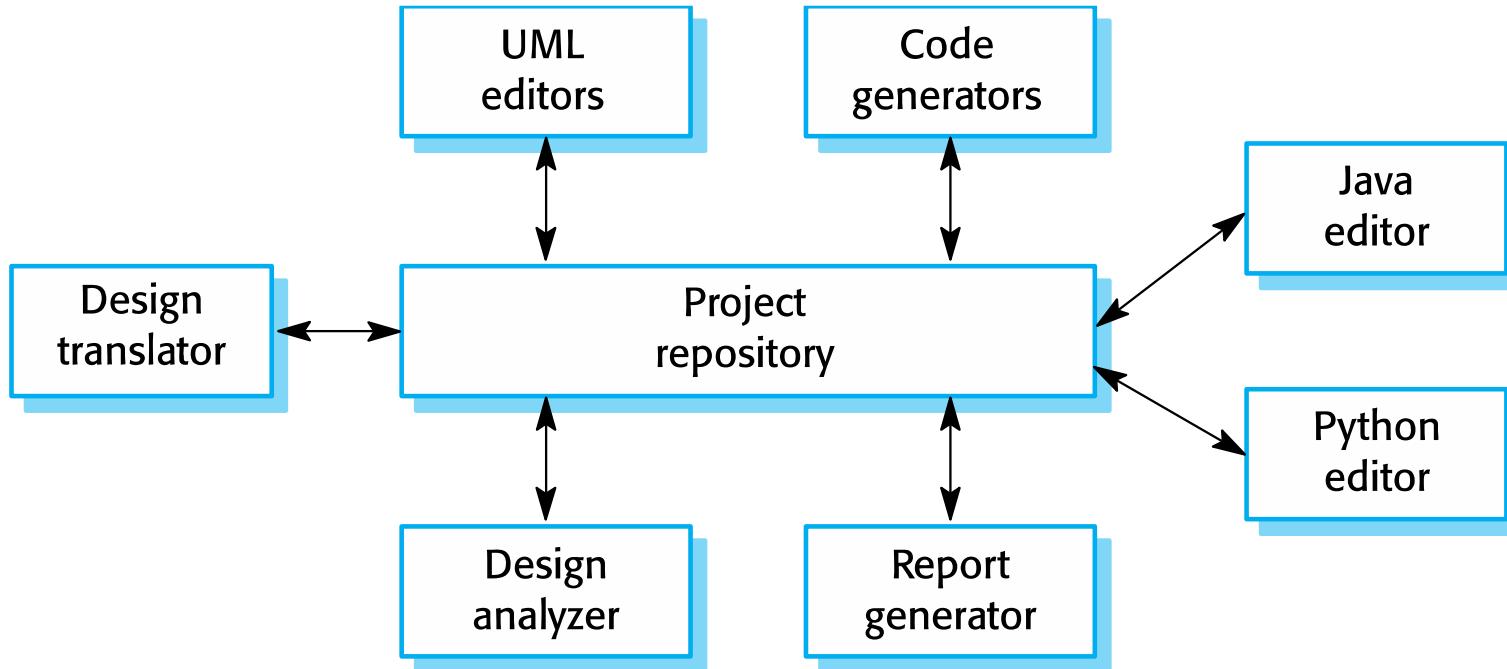
názov	Úložisko
Popis	Všetky dátá v systéme sú spravované v centrálnom úložisku, ktoré je prístupné všetkým komponentom systému. Komponenty neinteragujú priamo, iba prostredníctvom úložiska.
Príklad	Príklad IDE, kde komponenty používajú úložisko informácií o návrhu systému. Každý softvérový nástroj generuje informácie, ktoré sú potom dostupné na použitie inými nástrojmi.
Pri použití	Tento vzor by ste mali použiť, ak máte systém, v ktorom sa generujú veľké objemy informácií, ktoré je potrebné uchovávať na dlhú dobu. Môžete ho použiť aj v systémoch riadených údajmi, kde zahrnutie údajov do úložiska spúšťa akciu alebo nástroj.
Výhody	Komponenty môžu byť nezávislé – nemusia vedieť o existencii iných komponentov. Zmeny vykonané jedným komponentom sa môžu rozšíriť na všetky komponenty. Všetky dátá je možné spravovať konzistentne (napr. zálohovanie v rovnakom čase), keďže sú všetky na jednom mieste.
Nevýhody	Úložisko je jediným bodom zlyhania, takže problémy v úložisku ovplyvňujú celý systém. Môže ísť o neefektívnosť pri organizovaní všetkej komunikácie cez úložisko. Distribúcia úložiska na niekoľko počítačov môže byť náročná .

Centrálne úložisko pre IDE



Software Engineering
6th edition

Ian Sommerville



(1.3) Zdieľané služby a servery



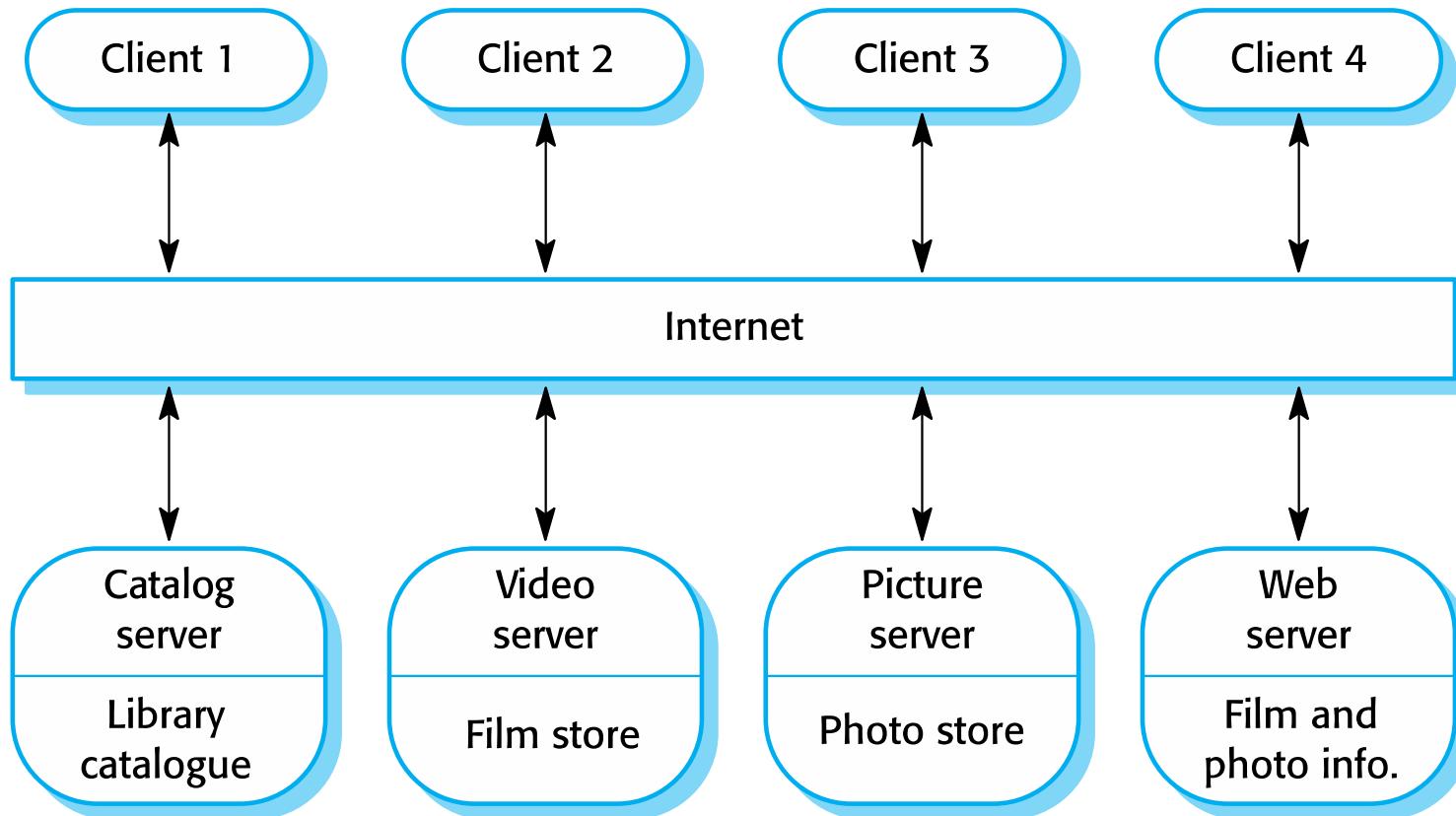
- ✧ Model distribuovaného systému, ktorý ukazuje, ako sú údaje a spracovanie distribuované v rámci rôznych komponentov .
 - Môže byť implementovaný na jednom počítači.
- ✧ Sada samostatných **serverov**, ktoré poskytujú špecifické služby, ako je tlač, správa údajov, atď.
- ✧ Skupina **klientov**, ktorí tieto služby využívajú.
- ✧ **Siet'**, ktorá umožňuje klientom prístup k serverom.

Vzor klient-server



názov	Klient- server
Popis	V architektúre klient-server je funkčnosť systému organizovaná do služieb, pričom každá služba je poskytovaná zo samostatného servera. Klienti sú používateľmi týchto služieb a pristupujú k serverom, aby ich mohli využívať.
Príklad	Príklad filmovej a video/DVD knižnice organizovanej ako systém klient-server.
Pri použití	Používa sa, keď je potrebné pristupovať k údajom v zdieľanej databáze z rôznych miest. Pretože servery možno replikovať, môžu sa použiť aj vtedy, keď je zaťaženie systému premenlivé.
Výhody	Hlavnou výhodou tohto modelu je, že servery môžu byť distribuované cez siet. Všeobecná funkcia (napr. tlačová služba) môže byť dostupná pre všetkých klientov a nemusí byť implementovaná všetkými službami.
Nevýhody	Každá služba je jediným bodom zlyhania, ktorý je náchylný na útoky odmietnutia služby alebo zlyhanie servera. Výkon môže byť nepredvídateľný, pretože závisí od siete, ako aj od systému. Ak servery vlastnia rôzne organizácie, môžu nastať problémy so správou .

Architektúra klient-server pre filmovú knižnicu





(2) Modulárne štýly rozkladu

- ✧ Štýly rozkladu (pod)systémov na moduly.
 - (Sub)systém je systém ako taký, ktorého prevádzka je nezávislá od služieb poskytovaných inými podsystémami.
 - Modul je komponent systému, ktorý poskytuje služby iným komponentom, ale za normálnych okolností by sa nepovažoval za samostatný systém.
- ✧ Pokryté dva modulárne modely rozkladu
 - **(2.1) Objektový model**, kde je systém rozložený na interagujúce objekty;
 - **(2.2) Model potrubia alebo toku údajov**, kde je systém rozložený na funkčné moduly, ktoré transformujú vstupy na výstupy.

(2.2) Architektúra potrubia a filtra



- ✧ Funkčné transformácie spracovávajú svoje vstupy a vytvárajú výstupy.
- ✧ Môže byť označovaný ako model potrubia a filtra (ako v prostredí UNIX).
- ✧ Varianty tohto prístupu sú veľmi bežné. Ked' sú transformácie sekvenčné, ide o **dávkový sekvenčný model**, ktorý sa vo veľkej miere používa v systémoch na spracovanie údajov.
- ✧ Nie je vhodný pre interaktívne systémy.

Vzor potrubia a filtra



názov	Potrubie a filter
Popis	Spracovanie údajov v systéme je organizované tak, že každý komponent spracovania (filter) je diskrétny a vykonáva jeden typ transformácie údajov. Dáta prechádzajú (ako v potrubí) z jedného komponentu do druhého na spracovanie.
Príklad	Príklad potrubného a filtračného systému používaného na spracovanie faktúr.
Pri použití	Bežne sa používa v aplikáciách na spracovanie údajov (dávkové aj transakčné), kde sa vstupy spracúvajú v samostatných fázach, aby sa vytvorili súvisiace výstupy.
Výhody	Lahko pochopiteľné a podporuje opäťovné použitie transformácie. Štýl pracovného toku zodpovedá štruktúre mnohých obchodných procesov. Evolúcia pridávaním transformácií je priamočiara. Môže byť implementovaný ako sekvenčný alebo súbežný systém.
Nevýhody	Formát prenosu údajov sa musí dohodnúť medzi komunikujúcimi transformáciami. Každá transformácia musí analyzovať svoj vstup a zrušiť analýzu výstupu do dohodnutej formy. To zvyšuje réžiu systému a môže znamenať, že nie je možné opäťovne použiť funkčné transformácie, ktoré používajú nekompatibilné dátové štruktúry .



(3) Štýly ovládania

- ✧ Zaoberajú sa riadiacim tokom medzi podsystémami. Na rozdiel od modelu rozkladu systému.
- ✧ (3.1) Centralizované ovládanie
 - Jeden podsystém má celkovú zodpovednosť za riadenie a spúšťa a zastavuje ďalšie podsystémy.
- ✧ (3.2) Ovládanie založené na udalostiach
 - Každý podsystém môže reagovať na externe generované udalosti z iných podsystémov alebo prostredia systému.



Centralizované ovládanie

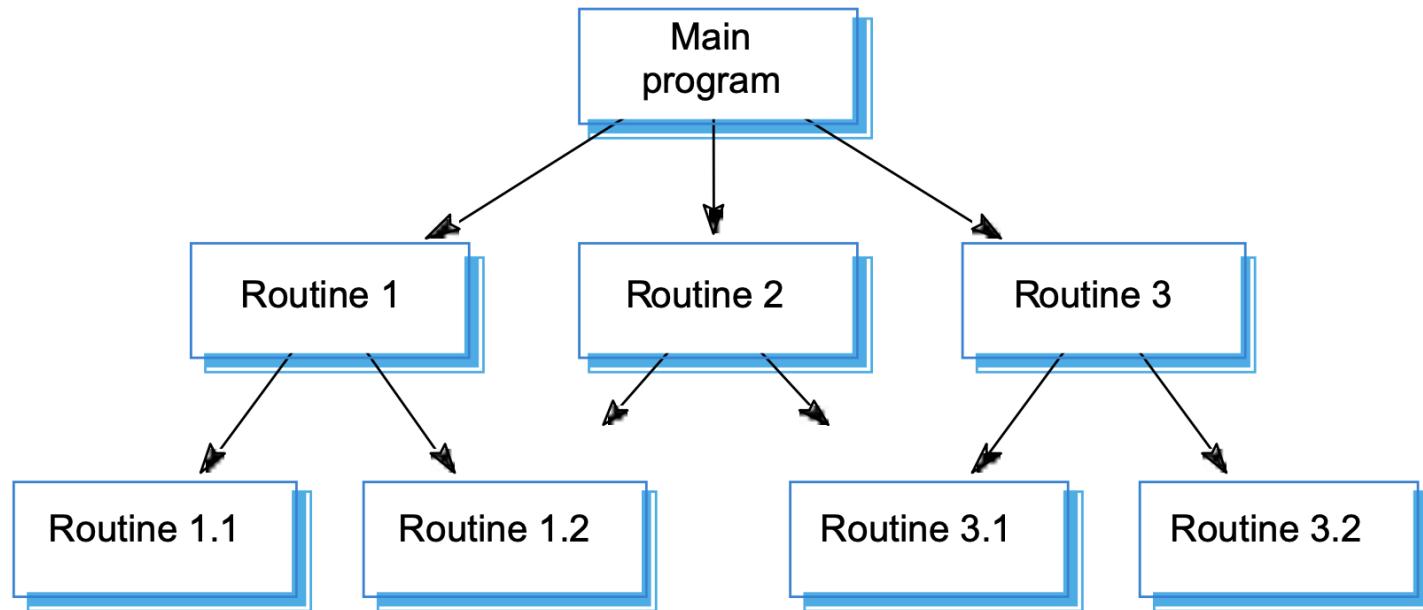
- ✧ Riadiaci subsystém preberá zodpovednosť za riadenie vykonávania iných subsystémov.
- ✧ (3.1.1) Model odovzdania a vrátenia riadenia
 - Model podprogramu zhora nadol, kde riadenie začína na vrchole hierarchie podprogramu a pohybuje sa smerom nadol. Použiteľné pre sekvenčné systémy.
- ✧ (3.1.2) Manažérsky model
 - Použiteľné pre súbežné systémy. Jeden systémový komponent riadi zastavovanie, spúšťanie a koordináciu ostatných procesov systému. Môže byť implementovaný aj v sekvenčných systémoch.

(3.1.1) Model odovzdania a vrátenia riadenia

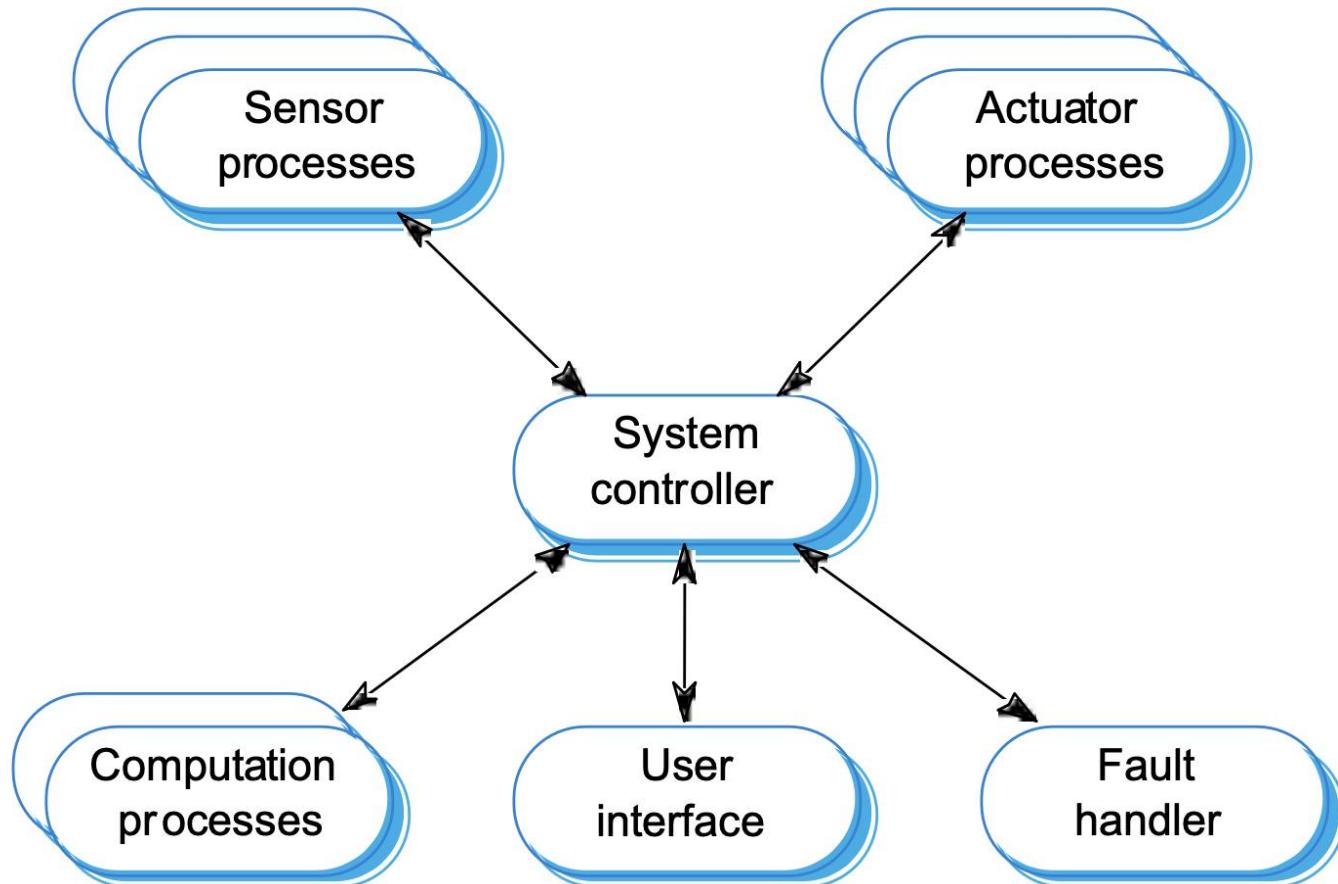


Software Engineering
6th edition

Ian Sommerville



(3.1.2) Manažérsky model



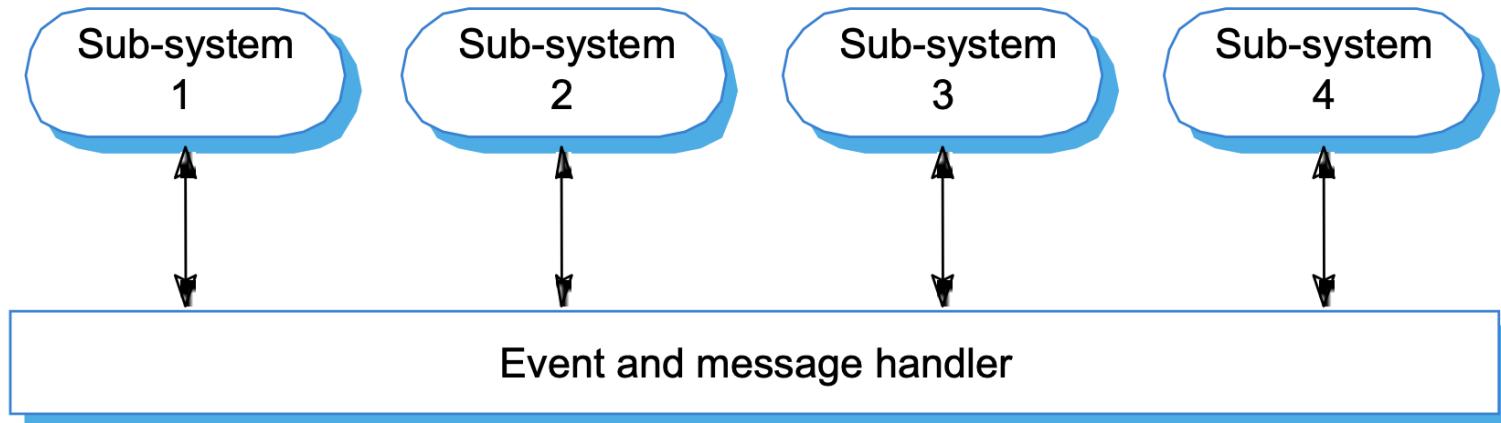
(3.2) Ovládanie založené na udalostíach



- ✧ Poháňané externe generovanými udalosťami, kde načasovanie udalosti je mimo kontroly podsystémov, ktoré udalosť spracúvajú.
- ✧ Dva hlavné modely riadené udalosťami
 - **(3.2.1) Vysielacie modely** . Udalosť sa vysiela do všetkých podsystémov. Môže tak urobiť akýkoľvek podsystém, ktorý dokáže spracovať udalosť;
 - **(3.2.2) Modely riadené prerušením**. Používa sa v systémoch v reálnom čase, kde sú prerušenia detegované obsluhou prerušenia a odovzdané nejakému inému komponentu na spracovanie.
- ✧ Medzi ďalšie modely riadené udalosťami patria tabuľky a produkčné systémy.

(3.2.1) Vysielací model

- ✧ Efektívne pri integrácii podsystémov na rôznych počítačoch v sieti.
- ✧ Podsystémy registrujú záujem o konkrétnu udalosť. Ked' k tomu dôjde, riadenie sa prenesie na podsystém, ktorý môže udalosť zvládnuť.
- ✧ Riadiaca politika nie je vložená do obsluhy udalosti a správy. Podsystémy rozhodujú o udalostach, ktoré ich zaujímajú.
- ✧ Podsystémy však nevedia, či a kedy sa udalosť spracuje.



(3.2.2) Riadenie prerušením



- ✧ Používa sa v systémoch v reálnom čase, kde je nevyhnutná rýchla reakcia na udalosť.
- ✧ Existujú známe typy prerušení s obsluhou definovanou pre každý typ.
- ✧ Každý typ je spojený s umiestnením pamäte a hardvérový prepínač spôsobí prenos do jeho obsluhy.
- ✧ Umožňuje rýchlu odozvu, ale zložité programovanie a náročné overenie.

Systém riadený prerušením

