Sběr a specifikace požadavků na SW

Pozn. metodologie, metodika = jak provést příslušnou fázi

Požadavky na sw – schopnost nebo vlastnost, kterou má sw mít

- aby jej uživatel mohl používat k vyřešení problému nebo dosažení svého cíle
- aby splnil podmínky stanovené smlouvou, standardem, smlouvou nebo jinou specifikací

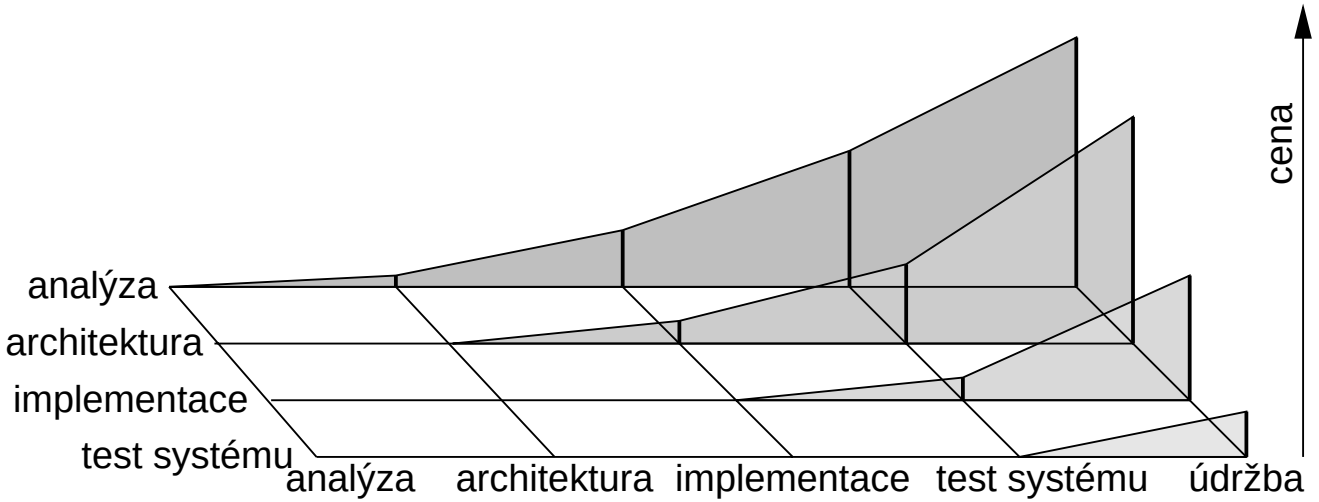
Tedy: požadavek není to, co uživatel nepotřebuje

- v této fázi definujeme:
  - jaké služby jsou od systému požadovány
  - jaká jsou omezení vývoje a výsledného produktu

- chyby v této fázi SW procesu vedou k problémům při návrhu a implementaci

Poznámka (cena změn):

změny v počátečních stadiích projektu jsou podstatně levnější (10x až 200x) než později (čím později defekt detekujeme, tím více času i peněz nás bude stát jeho oprava)



Např. defekt analýzy požadavků, jehož oprava by nás stála 1000 Kč během fáze analýzy požadavků, nás může stát 25 000 Kč během závěrečného testování systému (více u velkých systémů, méně u malých systémů).

Proto je zdravá snaha vyřešit problémy tak brzy, jak to jen jde.

Při specifikaci požadavků na rozsáhlé systémy obvykle **rozlišujeme 4 fáze**:

- studie proveditelnosti (feasibility study)
  - odhad, zda požadavky zákazníka mohou být splněny za pomocí existujících HW a SW technologií v rámci daných rozpočtových omezení
  - musí být rychlá a levná, výsledkem, zda přikročíme k podrobnější analýze

- sběr a analýza požadavků
  - zjišťujeme požadavky na nový systém pozorováním existujících systémů, diskusí s potenciálními uživateli a se zástupci zadavatele
  - může zahrnovat vývoj modelů a prototypů

- specifikace požadavků
  - výsledkem je dokument specifikace požadavků (DSP)
  - požadavky obvykle uvedeny ve dvou úrovních podrobnosti:
    - zákazník - vysokoúrovňový popis požadavků (user requirements) – musí jim rozumět
    - vývojáři - podrobnější specifikace systému (system requirements) – musí být schopni z ní vytvořit návrh systému

- validace požadavků
  - kontrola požadavků - zda jsou realistické, konzistentní a úplné
  - během této fáze nacházíme chyby v DSP => DSP musíme opravit
  - v průběhu této i předchozích fází se mohou objevit nové požadavky

Studie proveditelnosti
(v některých metodikách se překrývá s analýzou rizik, vizí projektu, stručným popisem projektu apod.)

- pro všechny nové systémy by SW proces měl začínat studií proveditelnosti
- snaží se zodpovědět následující otázky:
  - přispěje systém k celkovým záměrům organizace?
  - lze systém implementovat za pomocí existujících HW a SW technologií?
  - bude možné systém integrovat s již existujícími systémy?
  - bude vývoj systému finančně efektivní (tj. neproděláme na tom?), resp. lze systém realizovat v rámci daných rozpočtových omezení?

- vstupem je obecný popis + jakým způsobem má být v organizaci využíván
- výstupem je zpráva, která doporučuje nebo nedoporučuje pokračovat ve vývoji
- obecné fáze - identifikace potřebných informací, získání informací, vytvoření zprávy

Informační zdroje – lidé kolem zamýšleného systému – základní otázky:

- jaké jsou problémy se současným procesem zpracování informací, jak je má nový systém pomoci řešit?
- lze přenášet informace z a do jiných informačních systémů, které organizace již provozuje?
- co musí být v systému podporováno a co nemusí?
- je zapotřebí technologie, která ještě nebyla použita?

Výsledná studie pak:

- obsahuje doporučení, zda pokračovat ve vývoji
- může navrhnout změny rozsahu, rozpočtu a časového plánu systému
- může navrhnout další vysokoúrovňové požadavky na systém

Analýza domény a zjištění požadavků

- za pomocí zadavatele se seznamujeme s aplikační doménou a zjišťujeme, jaké služby má systém poskytovat

Můžeme přepokládat, že:

- zadavatel obvykle neví přesně, co od systému požaduje, resp. je pro něj obtížné to vyjádřit

- zadavatel může mít nerealistické požadavky, protože nezná pracnost (cenu) jejich realizace

- zadavatel vyjadřuje požadavky ve vlastních termínech a s implicitní znalostí vlastní práce (aplikační domény); vy je musíte pochopit

- různí zástupci zadavatele mají různé požadavky, které vyjadřují různým způsobem; vy musíte určit všechny zdroje požadavků, najít společné části a části, které jsou v konfliktu

- požadavky mohou ovlivňovat i „politické faktory“; například někteří manažeři mohou mít specifické požadavky, které by pomohly posílit jejich vliv v organizaci, vliv zainteresovaných hráčů (stakeholder)

- prostředí, ve kterém se provádí analýza, se průběžně mění – důležitost jednotlivých požadavků se může změnit; nové požadavky mohou přicházet od zástupců zadavatele, kteří do toho předtím neměli co mluvit.

Obecný model analýzy požadavků zahrnuje:

- porozumění aplikační doméně - analytik musí pochopit aplikační doménu např. pokud řeší informační systém pro autoservis, musí vědět jak fungují autoservisy
- sběr požadavků - analytik zjišťuje požadavky na systém od zástupců zadavatele
- klasifikace požadavků - nestrukturovanou množinu požadavků se snažíme logicky uspořádat
- řešení konfliktů - pokud je více zadavatelů, některé požadavky budou vzájemně v rozporu; rozpory požadavků musíme vyhledat a vyřešit
- určení priorit - konzultací se zadavatelem bychom měli určit nejdůležitější požadavky
- kontrola požadavků - musíme zjistit, zda jsou požadavky úplné, konzistentní a zda odpovídají tomu, co zadavatel od systému chce

Ve skutečnosti je vše opět iterativní proces, aneb…

Porozumění aplikační doméně

- velikost aplikační domény – čím větší aplikační doména, tím vyšší požadavky na analytika
- dosavadní zkušenost analytika – má už nějakou znalost domény (výhoda pro analytiky s širším rozhledem
- komunikační schopnosti analytika, schopnost abstrakce, nadhled
- umění pohybovat se v doméně a pozorovat postupy práce, toky informací apod.
- analytické vzory

- jednání se zadavatelem (zadavatel jedna osoba – malá organizace – velká organizace)
  - váš zákazník – váš pán (pozor: platí pro předmět obchodu, tj. software, rozhodně ne však v jiných situacích)
  - jste odborníci, a…také hlavně lidé!
  - první dojem
  - nenecháte se vmanipulovat do nevýhodné pozice (leckde to na vás budou zkoušet)
  - umíte říkat ne
  - umíte říkat já vám nerozumím
  - nevzbuzujete falešné naděje (neslibujete věci a funkčnosti vašeho sw, o kterých víte, že je nedokážete splnit)
  - jste zodpovědní za to, co děláte a říkáte
  - jste smířeni s faktem, že se nikdy nezavděčíte všem (a ani se o to nepokoušíte, neb dobře víte, že byste stejně širší okolí neuspokojili a navíc byste nebyli spokojení sami se sebou)
  - víte, že moudří pochybují a hlupáci tvrdí

Sběr požadavků

- zabývá se zdroji požadavků (celková motivace systému, doménové znalosti, zadavatelé systému, provozní prostředí, prostředí organizace) a způsobem jejich získávání
- z vyprávění zadavatele (+ dalších zainteresovaných lidí) vytáhnout nutné a relevantní informace pro návrháře a programátory
- příprava např. seznam (okruhy) otázek vs. kreativita a umění improvizace

- je obtížný, i když jsou zadavatelé dostupní a ochotní spolupracovat

- existence mnoha technik:
  - interview = předem připravený rozhovor
  - případy užití - definice případů použití se scénáři jejich průběhu
  - tvorba prototypů - papírové modely obrazovek, SW prototypy; pomůže uživateli lépe pochopit, jaká informace se od něj požaduje
  - pozorování prací u zákazníka, případně účast analytiků dodavatele na pracích u zákazníka (relativně drahé)
  - analýza existujícího SW systému

Interview

- předem připravený rozhovor, který vede moderátor (klade otázky, přiděluje slovo)
- trvání ne delší než 2 hodiny na setkání
- předem připravený scénář – probírané okruhy, jejich pořadí, nenásilné dodržování scénáře

Př. cílené interview má strukturu:

- moderátor shrne účel interview a jeho strukturu
- posloupnost otázek k jednotlivým bodům
- závěrečné shrnutí + ověření, že informace byly správně pochopeny

- užívání otevřených otázek:
  Co má systém řešit? Jak to řešíte nyní? Kdo bude uživatelem systému?

- příklady uzavřených otázek:
  Je 50 položek přiměřené množství? Potřebujete větší formulář?

vyhněte se:
- nechtějte po uživatelích popis příliš složitých aktivit
- otázkám začínajícím "Proč", mohou vyprovokovat obranný postoj

Případy užití (use-cases)

- pro většinu lidí je jednodušší zacházet s případy z reálného života než s abstraktním popisem
  - např. dokáží pochopit a okomentovat scénář toho, jak budou interagovat se SW systémem, využijeme toho

- metodika pocházející z RUP (Rational Unified Process)
  - vhodná pro střední a velké systémy
  - často se používá, je součástí UML a má podporu v CASE nástrojích

- na základě požadavků zákazníka najdeme aktéry a případy užití, stručně je popíšeme
- modelu rozumí zákazník
- pracovní setkání pro hledání případů užití (brainstorming, potřebujeme lidi různých znalostí a zkušeností, do 10 lidí), polovina vývojáři a polovina zástupci zákazníka, prostředníkem moderátor

Postup:

- identifikace aktérů
- identifikace případů použití
- vytvoření popisu pro každý případ použití
- popis toku událostí pro každý případ použití
- strukturování případů použití
- identifikace analytických tříd atd.

Blíže v objektové analýze, podrobně v ASWI

Pozn. některé projekty používají případy použití pouze neformálně pro sběr požadavků, požadavky ale zapisují a uchovávají jiným způsobem

Specifikace požadavků

- základní klasifikaci typů požadavků:
  - funkční,
  - mimofunkční
  - doménové

- uživatelské a systémové požadavky

Funkční požadavky (functional requirements, česky někdy také funkcionální požadavky)

- popisují funkce nebo služby, které jsou od systému očekávány
- například požadavky na univerzitní knihovní systém:
  - uživatelé mají možnost prohledávat databázi titulů knihovny nebo její podmnožinu
  - systém poskytuje uživatelům vhodné prohlížeče pro čtení dokumentů v úložišti dokumentů
  - uživatelé si mohou rezervovat titul knížky
  - uživatelé si mohou prohlédnout dokument vlastních výpůjček
  - systém umožní přidání nového čtenáře

Mimofunkční požadavky (non-functional requirements, česky někdy také nefunkcionální požadavky)

- netýkají se funkcí systému, ale vlastností jako je spolehlivost, propustnost, čas odpovědi a obsazené místo na disku nebo v paměti
- často kritičtější než jednotlivé funkční požadavky, např. pokud je řídící systém letadla nespolehlivý, je nepoužitelný
- týkají se SW procesu, např. v procesu musí být použit určitý standard pro řízení jakosti (ISO 9000), návrh musí být vytvořen určitým CASE nástrojem, pro implementaci, musí být použit daný programovací jazyk, produkt musí být dodán do určitého data apod.
- požadavky dané vnějšími faktory, např. legislativní požadavky (systém musí být navržen v souladu se zákonem na ochranu osobních informací apod.)

Příklady mimofunkčních požadavků:

- veškerá komunikace mezi uživatelem a systémem musí být vyjádřitelná ve znakové sadě ISO 8859-2.
  - definuje omezení návrhu uživatelského rozhraní, tj. není funkce, ale omezení => mimofunkční požadavek

- proces vývoje systému a všechny dokumenty mají odpovídat softwarovému procesu definovanému ve standardu XYZ.
  - mimofunkční požadavek týkající se SW procesu

- veškeré funkce systému budou dostupné minimálně 99% času v pracovních dnech (Po-Pá) od 8.00 do 18.00
  - mimofunkční požadavek na spolehlivost

- systém nebude operátorům systému poskytovat žádné osobní informace o zákaznících kromě jména a čísla zákazníka.
  - *požadavek daný vnějšími faktory, např. přijatelností systému pro veřejnost*

Maximální čas odezvy systému při maximálním definovaném zatížení (viz…) bude 0.3 s
- *mimofunkční požadavek na čas odezvy*

Časté problémy mimofunkčních požadavků :

- obecně definované mimofunkční požadavky jsou obtížně ověřitelné, takže mohou působit neshody po dodání systému zákazníkovi

  např. *Systém má být snadno použitelný a nebezpečí chyb uživatelů by mělo být minimalizováno.*

  Ověřitelná definice: *Zkušené účetní by měly být schopny používat všechny funkce systému po třídenním zaškolení. Počet chyb po zaškolení by neměl přesáhnout dvě za den.*

Mnoho požadavků je obtížné definovat měřitelným způsobem, někdy to ani nejde (např. udržovatelnost systému)

Mimofunkční požadavky mohou být v konfliktu, např.:

- aplikace bude přenositelná na operační systémy Windows a Unix
- aplikace bude umět pracovat se soubory ve formátu MS Word

je zapotřebí vyřešit – včas!

Příklad mimofunkčních požadavků a jejich metrik

- *rychlost* - průchodnost, např. počet transakcí za sekundu, čas odpovědi na požadavek uživatele nebo na událost (průměrný, maximální), čas překreslení obrazovky
- *velikost* - velikost v paměti, na disku, např. v KB
- *snadnost použití* - doba potřebná pro zaškolení, doba trvání typické úlohy, rozsah nápovědy, soulad se standardem pro GUI
- *spolehlivost* - počet havárií za časovou jednotku, průměrná doba mezi haváriemi, pravděpodobnost nedostupnosti, počet chyb/1000 řádek kódu
- *robustnost* - doba zotavení po havárii, doba opravy po havárii, pravděpodobnost poškození dat při havárii, přijatelné chování při degradaci systému
- *přenositelnost* - velikost kódu závislého na platformě (např. v %), počet cílových platforem
- *kapacita* - počet zákazníků, který systém zvládne

Doménové požadavky (domain requirements)

- vyplývají z aplikační domény, nikoli ze specifických požadavků zadavatele
- mohou být funkční nebo mimofunkční

- problémy s doménovými požadavky:
  - specialisté rozumí doméně natolik dobře, že je nenapadne doménové požadavky specifikovat explicitně
  - vývojáři o nich nemusejí vědět nebo jim rozumět

Rozdílné úrovně popisu - pro různé čtenáře:

- uživatelská specifikace požadavků (user requirements specification)
  - vysokoúrovňový popis funkčních a mimofunkčních požadavků zákazníka
  - musí být srozumitelné pro uživatele, kteří nemají technické znalosti (manažery klienta a dodavatele, koncové uživatele)

- systémová specifikace požadavků (system requirements specification)
  - podrobnější specifikace uživatelských požadavků pro vývojáře dodavatele
  - může sloužit jako základ kontraktu mezi zákazníkem a dodavatelem
  - slouží jako výchozí bod pro návrh systému

Příklad (uživatelská specifikace požadavků):

1. Systém musí poskytnout způsob reprezentace externích dokumentů a možnost jejich prohlížení.

Příklad (systémová specifikace požadavků):

1.1 Uživateli bude poskytnuta možnost definovat typy externích dokumentů.
1.2 Každý typ externího dokumentu bude na obrazovce reprezentován určitou ikonou.
1.3 Uživateli bude poskytnuta možnost definovat pro typ externího dokumentu vlastní ikonu.
1.4 Uživateli bude poskytnuta možnost sdružit typ externího dokumentu s prohlížečem.
1.5 Pokud uživatel vybere ikonu reprezentující externí dokument, výsledkem bude spuštění prohlížeče sdruženého s typem externího dokumentu pro dokument reprezentovaný vybranou ikonou.

Dokument specifikace požadavků

2 druhy dokumentů specifikujících požadavky:

1. **dokument specifikující požadavky na systém** (angl. Concept of Operations, ConOps document; používají se ještě další názvy)

- vysokoúrovňový popis požadavků z hlediska zadavatele (musí se vyjadřovat v termínech zadavatele, protože ho budou číst také zástupci zadavatele)
- kromě seznamu požadavků obsahuje informace o celkových záměrech systému, cílovém prostředí, omezeních, předpokladech a mimofunkčních požadavcích
- může obsahovat modely kontextu systému, případy užití, toky informací a práce apod.
- slouží pro validaci systémových požadavků

2. **dokument specifikující požadavky na software** (angl. Software Requirements Specification, SRS)

- v češtině termín Dokument specifikace požadavků (DSP)
- podrobná specifikace požadavků na software, odvozená z požadavků na systém
- předpokládá se, že čtenáři už mají ponětí o SW inženýrství => jazyk může být přesnější, notace podrobnější
- měl by obsahovat specifikaci uživatelských i systémových požadavků
  - ty mohou být sdruženy v jednom popisu
  - často lepší uvést uživatelské požadavky v úvodu k systémovým požadavkům
  - pokud by byl rozsah dokumentu neúměrný, je možné vytvořit systémové požadavky jako samostatné dokumenty
  - obsahuje oficiální vyjádření o tom, co se od vyvíjeného systému očekává => pro produkty vyvíjené na zakázku může sloužit jako základ kontraktu

Různé velké organizace definovaly vlastní standardy a doporučení pro strukturu obou typů dokumentů, např. IEEE std 1362-1998 (struktura ConOps) a IEEE std 830-1998 (struktura DSP).

Např. struktura podle IEEE/ANSI 830 (IEEE Guide to Software Requirements Specifications) vypadá zjednodušeně takto:

Table of Contents	// obsah
1. Introduction	// úvod
1.1 Purpose of the requirements document	// účel DSP
1.2 Scope of the product	// rozsah produktu
1.3 Definitions, acronyms and abbreviations	// definice, zkratky
1.4 References	// odkazy
1.5 Overview of the remainder of the document	// přehled zbytku DSP
2. General description	// obecný popis
2.1 Product perspective (independent/part of)	// kontext produktu
2.2 Product functions	// funkce produktu

2.3 User characteristics // charakteristiky uživatelů
2.4 General constraints // obecná omezení
2.5 Assumptions and dependencies // předpoklady a závislosti
3. Specific requirements (functional, non-functional and interface requirements) // specifické požadavky
.... // (není std. struktura)
4. Appendices // přílohy
5. Index // rejstřík

Ve skutečnosti bude informace v DSP záviset na vyvíjeném produktu a na modelu SW procesu, takže je nutné si obecný model přizpůsobit.

Ne všude se dokumentují a udržují požadavky na vyvíjený SW. Zejména v malých firmách se silnou vizí je správa požadavků často považována za zbytečnou režii. Při dalším vývoji produktu pak nutnost vyhodnocovat navrhované změny vede k potřebě zjistit původní požadavky, které určovaly vlastnosti produktu.

Doporučení: pro praxi si navrhněte vlastní formát a používejte ho pro všechny DSP - sníží se tím pravděpodobnost, že na něco zapomenete.

Poznámka – dokument DSP k dispozici

- osnova je odvozená ze standardu IEEE 830 (MS Word)
- soubor můžete využít - před odevzdáním z něj zrušte nápovědu
- rozumným způsobem si ho přizpůsobte
- body, které se netýkají vaší úlohy, můžete odbýt poznámkou typu **Není**
- popis funkčních požadavků by měl vykazovat nějakou strukturu

DSP by měl splňovat následující body (volně):

- specifikovat pouze externí chování systému, tj. vyloučit z DSP návrh SW do té míry, do jaké je to možné, někdy jsou ale požadavky a design neoddělitelné, např. systém může komunikovat s jiným systémem, z čehož vyplývají požadavky na design
- být strukturován tak, aby v něm bylo snadné provádět změny
  - popis lokalizovaný a volně vázaný
- specifikovat omezení implementace
- charakterizovat přijatelné odpovědi na nežádoucí události
- zaznamenat představu o životním cyklu systému

Způsoby specifikace požadavků

- přirozený jazyk
- formuláře
- případy užití
- pseudokódy a specifikace rozhraní

Přirozený jazyk

- obvyklý a nevyhnutelný popis požadavků - výhodou srozumitelnost pro uživatele i pro vývojáře

- také nevýhody
  - nejednoznačnost popisu
  - složité koncepce (např. algoritmy) je obtížné popsat přesně
  - příliš flexibilní - stejná věc se dá říci mnoha různými způsoby; jak čtenář zjistí, že se požadavky liší a čím?
  - neexistuje jednoduchý způsob modularizace - jak zjistíte důsledek změny požadavků, tj. kterých všech dalších požadavků se změna dotkne?

- nutnost používat jazyk jednoduše, snažit se vyhnout běžným příčinám chybné interpretace

Příklad problematické definice:
Pokud uživatel zadá jméno delší, než je šířka formuláře, jeho pokračování se zobrazí na následující obrazovce. (Nejednoznačné: zobrazí se na následující obrazovce pokračování jména, nebo formuláře?)

Vyhněte se:

- dlouhým souvětím s mnoha vedlejšími větami
- používání termínů s několika přijatelnými významy
- prezentaci několika požadavků jako jediného požadavku
- nekonzistenci v používání termínů, např. používání synonym.

měření kvality DSP
u velmi rozsáhlých projektů metriky měřící kvalitu DSP. Měřit lze

- velikost a čitelnost textu (délka vět apod.),
- strukturu textu (hloubku struktury a délka částí).

Slabá místa specifikace lze najít hledáním výskytu neurčitých slov nebo frází (několik, především) apod.

proto v systémových specifikacích snaha přidat strukturu omezením nejednoznačnosti např. formuláře nebo pseudokódy

Formuláře

- přirozený jazyk příliš flexibilní, někdy se přidává struktura pomocí formulářů
- jeden nebo více typů
- obsah:
  - popis specifikované funkce nebo entity
  - popis vstupů a odkud se berou
  - popis výstupů a kam putují
  - jaké další entity specifikovaná funkce nebo entita používá
  - případné vstupní a výstupní podmínky (pre-conditions a post-conditions), tj. co platí při vstupu do funkce a co při výstupu z ní
  - pokud vznikají vedlejší efekty, pak jejich popis

Příklad systémové specifikace funkce pro vkládání prvku do diagramu, zapsaná ve formě formuláře:

Funkce: Vlož prvek do diagramu.

Popis: *Vloží prvek do existujícího diagramu. Uživatel určí typ prvku a jeho pozici.*

Vstupy: *Typ prvku, Pozice prvku, Identifikátor diagramu.*

Zdroje: *Typ prvku a Pozici prvku zadá uživatel, Identifikátor diagramu získáme z databáze diagramů.*

Výstupy: *Identifikátor diagramu.*

Úložiště: *Databáze diagramů. Při dokončení operace je proveden COMMIT.*

Vyžaduje: *Diagram odpovídající vstupnímu Identifikátoru diagramu.*

Vstupní podmínka: *Diagram je otevřen a zobrazen na obrazovce uživatele.*

Výstupní podmínka: *Diagram je nezměněn kromě přidání prvku určeného typu na určenou pozici.*

Vedlejší efekty: *Nejsou.*

Případy užití

- viz dříve, později a ASWI
- popis kontextu systému a popis funkčních požadavků
- grafická notace UML pro popis objektově orientovaných modelů systému

Pseudokódy

- někdy je funkce specifikována jako posloupnost jednodušších akcí, pořadí je **podstatné**
- v přirozeném jazyce bychom obtížně vyjadřovali např. vnořené podmínky nebo smyčky
- vhodné doplnit specifikaci popisem v pseudokódu

vlastnosti:

- jazyk s abstraktními konstrukcemi, které právě potřebujeme (sekvence jednoduchých příkazů, iterace, větvení):
- vnoření konstrukcí je vyjádřeno odsazením
- vyhýbáme se syntaktickým konstrukcím cílového programovacího jazyka (které by nás zbytečně omezovaly a sváděly k programování), popisujeme požadovaný záměr, nikoli jak bude v cílovém jazyce implementován
- musí umožňovat téměř automatickou konverzi do kódu (pokud příliš vysoká úroveň, nemusíme postřehnout problémy ve specifikaci)

Příklad (část popisu činnosti bankomatu):

Přečti kartu
Vypiš výzvu: Prosím zadejte PIN
Přečti zadané_PIN
Opakuj nejvýše 3x
    Přečti zadané_PIN
    Jestliže zadané_PIN je PIN_karty, pak opusť smyčku
Jestliže zadané_PIN není PIN_karty pak ...

Formuláře vs. pseudokód

- formuláře - celková specifikace systému
- pseudokód - řídící sekvence, rozhraní

Specifikace rozhraní

- pokud systém komunikuje s dalšími systémy, musí být přesně specifikováno softwarové nebo komunikační rozhraní
- specifikace rozhraní měla by být součástí DSP, pokud je rozsáhlá, tak v příloze; měla by být vytvořena brzy

2 typy rozhraní, které musí být definovány:

- procedurální rozhraní - existující podsystémy poskytují množinu služeb, které jsou přístupné prostřednictvím volání procedur rozhraní (např. *popis knihovních procedur nebo tříd programovacího jazyka, v klasických jazycích např. prototyp procedury nebo fce*, *vstupně/výstupní parametry, popis činnosti, návratová hodnota, v objektových jazycích např. obecný popis třídy, popis konstruktorů, popis metod)*

- popis předávaných dat - popis struktury dat - pro popis se používají datové modely (ERA diagramy) + popis reprezentace dat, např. *datum je reprezentován jako řetězec*

další možnosti vyjádření specifikace požadavků

- grafické notace - např. ostatní diagramy UML
- formální (matematická) specifikace (např. konečný automat)

Klasifikace a uspořádání požadavků

- množina požadavků nestrukturovaná -> snaha o uspořádání
- klasifikace dle různých kritérií, např.:

  - požadavky na funkce a mimofunkční požadavky (viz výše)
  - podle priority, např. na nutné, žádoucí, vhodné, volitelné (často je zapotřebí vzít v úvahu cenu vývoje)
  - podle rozsahu; např. některé mimofunkční požadavky jsou globální, zatímco některé požadavky na funkce je možné změnit, aniž by měly vliv na ostatní fce systému
  - podle pravděpodobnosti změny na trvalé a nestálé požadavky; pro vývojáře může být snazší reagovat na změnu, pokud v DSP označíme požadavek jako nestálý.

Příklad nestrukturovaného požadavku:

2.2.15 *Zobrazování mřížky.*
2.2.15.1 *Aby bylo možné umísťovat entity do diagramu, uživatel může zapnout zobrazování mřížky buď v centimetrech nebo v palcích, a to pomocí volby na řídícím panelu. Na počátku se mřížka nezobrazuje. Mřížka může být vypnuta a zapnuta kdykoli během relace a kdykoli může být přepnuta mezi centimetry nebo palci. Možnost zobrazovat mřížku bude i pro zmenšené pohledy, ale počet řádků mřížky bude omezen tak, aby mřížka při prohlížení zmenšeného obrázku nerušila.*

První věta obsahuje 3 typy požadavků:

1. Koncepční funkční požadavek - editor má poskytovat mřížku.
2. Mimofunkční požadavek - jednotky mřížky.
3. Mimofunkční požadavek na UI - mřížku bude zapínat a vypínat uživatel.

další nedostatky:

- neúplnost - chybí inicializační informace - jakou jednotku bude mřížka používat po zapnutí?
- Smísení uživatelských a systémových požadavků
- Nevydělené zdůvodnění požadavků

Příklad uživatelské specifikace - přepsání s vynecháním detailů:

2.2.15 *Mřížka editoru*.

2.2.15.1 *Editor bude poskytovat možnost zobrazit mřížku, tj. matrici horizontálních a vertikálních čar zobrazených na pozadí editoru. Mřížka bude pasivní a určování pozice jednotlivých elementů bude záležitostí uživatele.*

Zdůvodnění: Mřížka uživateli pomůže lépe rozmisťovat elementy diagramu. I když aktivní mřížka (tj. taková, kde se elementy "přilepují" k čarám mřížky) může být užitečná, je takové umisťování nepřesné. Umístění entit určí nejlépe uživatel.

zdůvodnění důležité - pokud dojde ke změně požadavků, můžeme určit, co vedlo k původnímu požadavku

Detekce a řešení konfliktů

- konflikty nastávají např.
  - pokud dva uživatelé požadují vzájemně neslučitelné vlastnosti
  - existuje rozpor mezi požadovanými schopnostmi a danými omezeními
- ve většině případů není vhodné, aby rozhodli vývojáři (konkrétní rozhodnutí - zástupce zadavatele

Validace požadavků

- vstupem úplný DSP
- zjišťujeme, zda jsou požadavky úplné, konzistentní a zda odpovídají tomu, co zadavatel od systému chce

u požadavků se kontroluje:

- *platnost*
  - uživatel si myslí, že systém má poskytovat určité funkce, ale podrobnější analýza může ukázat něco jiného
  - různí uživatelé mají různé požadavky, budou kompromisy platné?

- *konzistence* - požadavky v dokumentu nesmějí být v konfliktu, tj. nesmějí být různé popisy nebo různá omezení stejné fce

- *úplnost* - definovány by měly být všechny fce a omezení systému

- *realizovatelnost* - systém je implementován s danými prostředky a v daném čase

- *ověřitelnost* - systémové požadavky jsou napsány tak, že jsou ověřitelné (ušetříme si dohadování se zákazníkem při předávání produktu)

- *sledovatelnost* původu požadavku - abychom dokázali odhadnout dopad změny

Použitelné metody:

- přezkoumání (reviews)
- prototypování
- tvorba testů
- automatická analýza konzistence.

Přezkoumání (reviews)

- požadavky jsou systematicky zkontrolovány týmem (manuální proces, více čtenářů od zákazníka i od dodavatele)
- vývojový tým provází klienta systémovými požadavky, vysvětluje důsledky každého požadavku, kontroluje konzistenci, úplnost atd.

Prototypování

- zákazníkovi předvedeme spustitelný model systému, může zjistit, zda odpovídá požadavkům (typicky chování uživatelského rozhraní)
- nevýhoda prototypů - pozornost uživatele odvádějí kosmetické záležitosti nebo omezení prototypu (proto někdy papírové modely obrazovek)

Generování testovacích případů

- vytvořený test požadavků často odhalí problémy
- pokud je obtížné vytvořit test, bude požadavek obtížně implementovatelný

Automatická analýza konzistence

- pro modely ve formální nebo strukturované notaci ???

Správa požadavků

- požadavky na velký systém se neustále mění -> nutný proces řízení změn systémových požadavků
- trvalé a nestálé požadavky
    - trvalé požadavky - relativně stabilní, odvozeny ze základní funkce organizace nebo z aplikační domény (např. v nemocnici vždy pacienti, lékaři a sestry)
    - nestálé požadavky -. pravděpodobně se změní během vývoje nebo po uvedení systému do provozu (např. nemocnice - pravděpodobně se změní systém plateb od zdravotních pojišťoven)

Správa požadavků začíná plánováním, rozhodne se:

- způsob identifikace požadavků - každý požadavek bude mít jedinečný identifikátor, abychom ho mohli odkazovat (křížové reference apod.)
- proces změny požadavků - definujeme proces, abychom se ke změnám požadavků chovali konzistentním způsobem - viz níže
- sledovatelnost - které vztahy mezi požadavky navzájem atd. budeme uchovávat a jak (čím více, tím dražší)
    - zdroj požadavku - kdo požadavek navrhl
    - vztahy mezi požadavky v DSP; pro určení kolika požadavků se změna dotkne
    - vztahy mezi požadavky a designem systému; pro určení dopadu změny na systémový design a implementaci
- jaké nástroje se použijí pro uchovávání informací o požadavcích (malé projekty - postačují prostředky jako textové a tabulkové procesory, databáze; velké projekty - CASE nástroje)

Sledovatelnost požadavků (traceability)

- znamená možnost sledovat požadavky zpět k jejich zdroji

matice závislostí požadavků - mapuje např. vzájemné závislosti požadavků (pro malé množství požadavků, jinak její údržba drahá)

- prvky - jak závisí požadavek v řádku na požadavcích daných sloupcem
- U (Uses) - požadavek v řádku používá možnosti dané požadavkem ve sloupci
- R (Relates) - nějaký slabší vztah, např. oba části stejného podsystému

Id	1.1	1.2	1.3	1.4	1.5
1.1	-	U	R	-	-
1.2	-	-	U	-	-
1.3	R	-	-	-	-

databáze požadavků - součást CASE nástrojů pro správu požadavků (tyto nástroje často umí vygenerovat matici nebo graf závislostí, graficky znázornit propagaci změn, generovat zprávy o stavu požadavků, generovat DSP podle zvoleného standardu apod.

Proces změny požadavků

- všechny navrhované změny požadavků by měly podléhat procesu o třech základních krocích:

  - analýza problému a specifikace změny
  - analýza změny a určení její ceny (ve výsledku přijdeme za zákazníkem: stálo by to X, budete to chtít teď nebo později?)
  - implementace změny (ve výsledku modifikace DSP, případně designu a implementace)

urgentní požadavek na změnu -> tlak na provedení změny nejdříve v implementaci, pak zpětná modifikace DSP. Vede k tomu, že se DSP a implementace rozejdou: na začlenění změny do DSP se buď zapomene, nebo je DSP změněn nekonzistentně se změnou implementace.

Máme hotovou specifikaci požadavků

- V malých systémech můžeme z DSP pokračovat přímo tvorbou návrhu systému.
- Pokud je systém rozsáhlý, jsou zapotřebí např. další cykly spirály (spirálový model), které přidávají další podrobnosti a interpretují doménové požadavky pro vývojáře (aby byli schopni doménové požadavky správně interpretovat).

Poté je vývoj předán návrhářům.