Softwarové inženýrství

4. přednáška

Radek Janoštík

Univerzita Palackého v Olomouci

16. 10. 2025

Outline

- Požadavky – funkční, nefunkcionální, doménové
- Proces specifikace požadavků
- Doporučená četba

Požadavky – úvod

- = Popis služeb (a jejich podmínek) poskytovaných systémem
  - Zjišťování, analýza, dokumentování a ověření požadavků $\Leftrightarrow$ *requirements engineering*
- Více pohledů na *požadavek*
  - Abstraktní tvrzení o poskytované službě (výběrová řízení)
  - Komplexní formální definice funkcí systému (konkrétní systém)
- Různé typy požadavků od/pro různé publikum $\Rightarrow$ různá míra detailnosti
- *Uživatelské požadavky* – tvrzení v přirozeném jazyce, doprovázené diagramy
  - Jaké služby bude systém poskytovat za jakých podmínek
  - Manažeři, technici zákazníka, koncoví uživatelé, manažeři vývojářů, systémoví architekti
- *Systémové požadavky* – detailní popis jednotlivých funkcí a vlastností systému
  - Přesný, jasný popis všeho, co má být implementováno, součástí smlouvy
  - Koncoví uživatelé, technici zákazníka, systémoví architekti, vývojáři

Požadavky – příklad

- Mějme knihovní systém *LIBSYS*
- Uživatelský požadavek: *LIBSYS bude spravovat všechna data požadovaná autorskými licenčními agenturami z Velké Británie a ostatními.*
  - Pouze abstraktní tvrzení o poskytované funkcionalitě.
- Systémový požadavek:
  1. *Při vytváření požadavku na dokument z LIBSYS musí žadatel vyplnit požadavkový formulář s detaily o žadateli a požadavku.*
  2. *Požadavkové formuláře budou uloženy v systému po dobu pěti let od data požadavku.*
  3. *Všechny požadavkové formuláře musí být indexovány podle uživatele, jména materiálu a zpracovatele požadavku*
  4. *LIBSYS by měl udržovat záznamy o všech provedených požadavcích*
  - Detaily, zdůvodnění služeb a funkcionalit

Funkční požadavky

- Jaké služby by měl systém poskytovat
- Jak by měl systém reagovat na určité vstupy a výstupy, jak by se měl chovat
- Někdy obměnou – jak by se systém neměl chovat
- Definice by měla být:
  - *úplná* – všechny použité termíny a poskytované služby by měly být definované. Nezaměnitelné.
  - *konzistentní* – neměla by být v rozporu s definicí jiného požadavku
- U velkých systémů bývá problém obojí, jak spory vzniknou?
  - Odlišné skupiny uživatelů mají odlišné požadavky
  - Politická motivace
  - Jak vyřešit spor?

Funkční požadavky – příklady

1. Uživateli bude umožněno vyhledávat z celé *výchozí množiny databází* nebo si zvolí jejich podmnožinu.
   - Dříve by měla být definována *výchozí množina databází*
2. Systém bude poskytovat uživatelům *vhodné prohlížeče* pro čtení dokumentů z dokumentového úložiště.
   - Patrně zamýšleno „každý formát si bude moci uživatel prohlédnout“
   - Vývojáři ve spěchu implementují prohlížeč pro jeden formát a mohou tvrdit, že je to *vhodné*
3. Ke každé objednávce bude přiřazen unikátní identifikátor ORDER_ID, který si uživatelé budou moc zkopírovat do svého trvalého úložiště.
   - Podstatně konkrétnější než předchozí.

Nefunkcionální požadavky

- Nejsou přímo spojené s konkrétní funkcionalitou
- Požadavky na vlastnosti systému jako celku, často popis emergentních vlastností
- Často kritičtější než funkční (uživatel si „najde cestu“ $\times$ certifikace)
- Mohou se týkat i SW procesu jako takového
  - Rozpočet, legislativa
  - Geografické omezení, NDA
  - ISO 9000, jazyk, technologie
- Často vágně formulované $\Rightarrow$ obtížně ověřitelné
  - *Systém bude snadno použitelný a bude minimalizovat možné chyby uživatele*
  - vs. *Zkušený účetní bude schopen používat veškerou funkcionalitu systému po dvoudenním zaškolení, počet chyb po zaškolení nepřesáhne 2 za den.*
- Opět možný spor: *Velikost systému řízení sondy nesmí přesáhnout 8MB paměti a musí být napsán v jazyce Java™.*

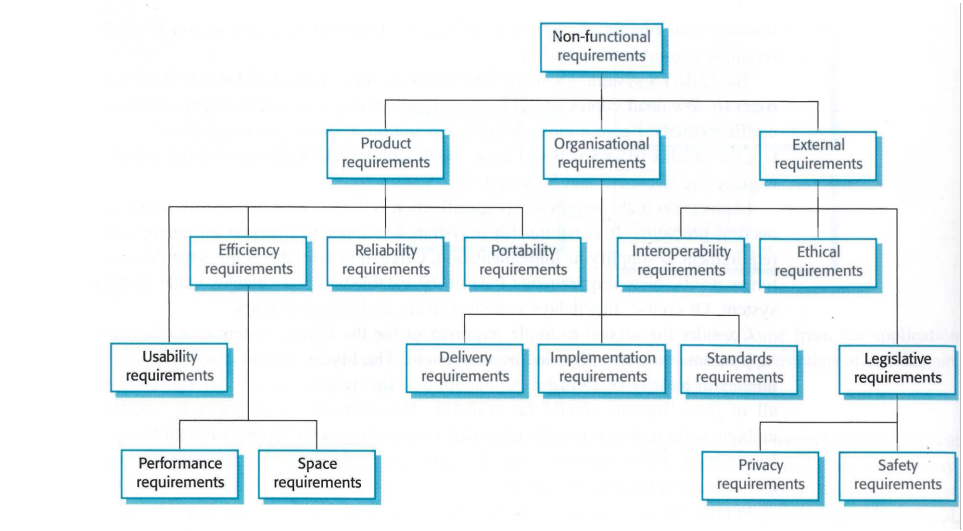
Nefunkcionální požadavky – rozdělení

1. *Požadavky na produkt* – specifikují přímé chování produktu
   - Požadavky na výkon, paměťovou náročnost
   - Spolehlivost, uživatelská přívětivost
2. *Organizační požadavky* – reflektují strukturu a pravidla zákazníka (ale i dodavatele)
   - Použité standardy v procesu
   - Programovací jazyk, metody návrhu
   - Termíny a podmínky dodání
   - Tvar dokumentace
3. *Externí požadavky* – vzešlé z použití a interakce s externími systémy
   - Jak budou systémy spolupracovat
   - Legislativa
   - Etické požadavky (bude přijato uživateli? Společností?)

Nefunkcionální požadavky – měřitelnost

- Formulace nefunkcionálních požadavků musí být *ověřitelná* – výroky by měly být kvantifikované
  - Konkrétní metrika – měřitelnost
- Jak měřit/kvantifikovat:
  - *Rychlost* – zpracované transakce/s, Uživatelská reakční doba, FPS, ...
  - *Velikost* – Megabajty, Gigabajty na disku, velikost použité RAM, ...
  - *Snadnost použití* – Školící doba, délka nápovědy, „učící křivka“, ...
  - *Spolehlivost* – Střední doba mezi poruchami, dostupnost, četnost poruch, ...
  - *Robustnost* – Čas pro restart po pádu, pravděpodobnost poškození dat, ...

Nefunkcionální požadavky – rozdělení



Doménové požadavky

- Požadavky určené přímo aplikační doménou
  - Bývá problém jim porozumět – neznalost domény
  - Aktéři procesu zjišťování na ně zapomínají (samozřejmost)
  - Funkční i nefunkcionální
- Jejich nesplnění $\Rightarrow$ kritické pro úspěch a celkovou ne/funkčnost systému
- Např.: *Uživatelské rozhraní systému pro všechny databáze musí být založeno na (knihovním) standardu Z39.50.*
- Zpomalení vlaku bude vypočteno jako $D_{vlaku} = D_{control} + D_{\delta}$, kde $D_{\delta} = a_g \cdot \frac{stoupani}{\alpha}$ a α je známa pro rozdílné typy vlaků

Uživatelské požadavky

- Opakování z předchozích slidů:
- *Uživatelské požadavky* – tvrzení v přirozeném jazyce, doprovázené diagramy
  - Jaké služby bude systém poskytovat za jakých podmínek
  - Manažeři, technici zákazníka, koncoví uživatelé, manažeři vývojářů, systémoví architekti
- Pojďme si o nich říci více
- Funkční a nefunkcionální požadavky, kterým rozumí uživatelé systému bez technických znalostí
- Popis pouze externího chování systému $\Rightarrow$ co nejméně návrhových specifik
  - Např. Uživatele nemusí zajímat v jakém jazyce je systém implementován.
  - Nepotřebuje vědět, na kolika pevných discích v RAID poli je databáze uložena.
- Měly by být psány v jednoduchém jazyce, bez programátorského žargonu
  - **Jednoduché** tabulky, formuláře, diagramy

Uživatelské požadavky

- Přirozený jazyk ale přináší řadu problémů
  1. *Nízká srozumitelnost* – bývá složité jednoduše a jednoznačně popsat požadavky, aniž bychom psali dlouhé a náročné věty
  2. *Míchání typů požadavků* – jasně by měly být rozlišeny funkční a nefunkcionální požadavky, cíle systému a návrhové informace
  3. *Slučování požadavků* – jednotlivé požadavky by měly být jasně odděleny
- Příklad: *LIBSYS bude poskytovat finanční účetní systém, který bude spravovat všechny záznamy o platbách, které uživatelé uskutečnili. Manažeři systému budou moci nastavit systém tak, že pravidelní uživatelé budou moci dostávat slevy.*
  - Je zde nějaký problém?
  - Druhá věta by měla být použita „až v“ systémových požadavcích nebo rozdělit
- V uživatelských požadavcích by nemělo být příliš mnoho technických detailů

Uživatelské požadavky – příklad

Požadavek 1 (Funkce mřížky)

Pro snadnější umístění entit v digramu si uživatel bude moci zapnout mřížku v centimetrech či palcích pomocí nastavení v ovládacím panelu. Ve výchozím stavu je mřížka vypnuta. Mřížka může být zapnuta kdykoliv během úprav a kdykoliv může být přepínána mezi centimetry a palci. Počet zobrazených čar bude omezen tak, aby nebylo v diagramu příliš mnoho čar.

- Je *Požadavek 1* dobře formulován?
- Míchá tři druhy požadavků
  - Koncepční, funkční požadavky – bude k dispozici mřížka a zdůvodnění
  - Systémový požadavek – detailní informace (jednotky)
  - Systémový požadavek – definice uživatelského rozhraní (jak zapínat)
- Přiměřená míra svobody pro implementaci (výhody i nevýhody)

Uživatelské požadavky – doporučení

- Standardní formát – pevně daná forma, které se budeme držet celou dobu
  - Zdůvodnění – proč je daný požadavek potřeba
  - Zdroj požadavku – s kým diskutovat změny
- Konzistentní používání jazyka – rozlišování povinné a volitelné funkcionality
  - Povinné – slova typu *musí poskytovat, bude poskytovat*
  - Nepovinné(ale žádoucí) – *by měl poskytovat*
- Rozlišení částí různým řezem písma
  - Tučná věta vystihující „to hlavní“
  - Nové pojmy kurzívou

Uživatelské požadavky – příklad

Požadavek 2 (Funkce mřížky)

Editor bude poskytovat funkci mřížky, při které bude zobrazena matice s vertikálními a horizontálními čarami na pozadí editoru. Mřížka by měla být pouze pasivním elementem, zarovnání elementů bude ponecháno na uživateli.

> *Zdůvodnění:* Mřížka pomáhá uživatelům vytvářet pěknější diagramy s dobrým uspořádáním prvků. „Přichytávání“ elementů k mřížce sice může být užitečné, ale chceme nechat uživateli absolutní svobodu.

Zdroj: Radek Janoštík, Softwarový Inženýr UPOL

Systémové požadavky

- Rozšíření uživatelských požadavků – určeno pro SW inženýry, pro návrh
- Přidání podrobností, jak přesně má být uživatelský požadavek splněn
- Často součástí smluv $\Rightarrow$ kompletní a konzistentní specifikace celého systému
- Stále by neměly obsahovat detaily z návrhu systému
  - Většinou se tomu ale již nevyhneme
  - Systémová architektura může pomoci strukturovat požadavky
- Opět psány v přirozeném jazyce $\Rightarrow$ problémy, ale „umí přečíst“ každý
  - Např. zápis algoritmů
  - Používat jednoduchý jazyk, který omezí možnosti nedorozumění – krátké věty
  - Konzistentnost při používání synonym
- Doplnění vhodnými diagramy, tabulkami, vzorečky
  - „Jeden **vhodný** obrázek řekne více než 1000 slov“

Systémové požadavky – strukturovaný přirozený jazyk

- Flexibilitu přirozeného jazyka je potřeba trochu zkrotit
  - Vynucená struktura pro formulování systémových požadavků
  - Neměnná, pro každý požadavek stejná
- Idea: Ponecháme si vyjadřovací sílu a srozumitelnost přirozeného jazyka, ale dáme jasná omezení strukturou.
- Definice několika typů formulářů/šablon, mělo by být obsaženo:
  1. *Funkce* – název požadavku a popis funkcionality či specifikované entity
  2. *Vstupy* – popis vstupů a jejich původ
  3. *Výstupy* – popis zamýšlených výstupů a jak budou využity
  4. *Závislosti* – které entity jsou pro funkcionality potřeba
  5. *Akce* – popis samotné provedené akce
  6. *Vstupní podmínky* – které musí být splněny před akcí
  7. *Výstupní podmínky* – které musí být splněny po akci
  8. *Vedlejší efekty* a jejich popis
- Taková struktura jasně vymezí úseky, ke kterým by se měl přirozený jazyk vyjadřovat

Strukturovaný přirozený jazyk – ukázka

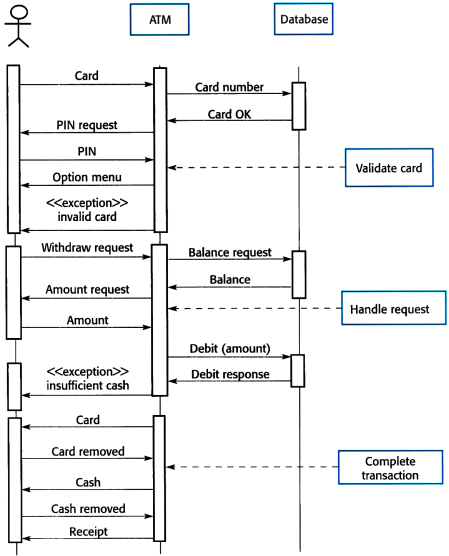
Požadavek 3 (Vložení prvku do diagramu)

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 efekt:	Žádný.

Strukturovaný přirozený jazyk

- Některé akce je velmi těžké popsat v několika větách (mnoho aktérů, stavů, podmínek)
- Je možné strukturovaný jazyk doplnit tabulkami se soupisem podmínek
- Velmi často se používá sekvenční diagramy z UML
- Funkční požadavek je poté mnohem přehlednější a uchopitelnější pro všechny strany

Sekvenční diagram



Specifikace rozhraní

- Navrhovaný systém většinou nebude existovat „ve vakuu“
- Komunikace/spolupráce s existujícími/budoucími systémy
- Rozhraní by mělo být specifikováno „co nejdřív“ a přesně (součástí DSP)
- Typy rozhraní:
  1. *Procedurální rozhraní* – poskytované služby pomocí volání procedur – API
  2. *Datové struktury*, které jsou posílány mezi (sub)systémy (UML)
  3. *Reprezentace dat* – v jaké formě jsou data předávána
     - Pořadí bitů, význam
     - XML, json
- Finální dokumentaci lze generovat z kódu, ale zatím pouze specifikujeme

Dokument specifikace požadavků (DSP)

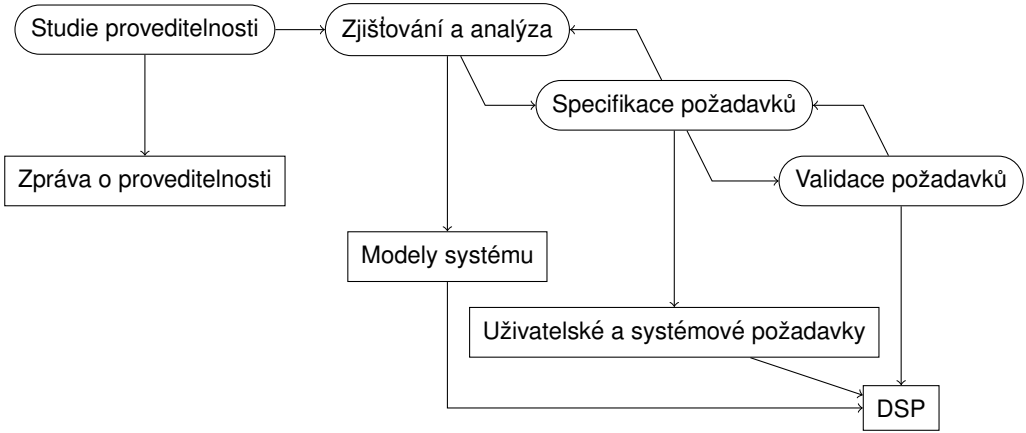
- Oficiální dokument – co mají vývojáři implementovat
- Uživatelské(úvod) i systémové požadavky(hlavní část)
- Jazyk může být trochu složitější, cílová skupina jsou SW inženýři
  - Ale i manažeři obou stran
- Formát závislý na modelu SW procesu (vodopád $\times$ evoluční/iterativní vývoj)
- V některých případech vůbec nemusí existovat (in-house vývoj, malé firmy, startupy)
  - Při „přerostení“ projektu lepší dopsat zpětně než lepit kód bez specifikace
- Může být velmi dlouhý $\Rightarrow$ důraz na dobrou logickou strukturu, odkazy, popisky, ...

DSP podle IEEE/ANSI 830

- Velké organizaci si definují vlastní standardy a doporučení pro DSP, kterých se pevně drží
- Například IEEE/ANSI 830 – *IEEE Guide to Software Requirements Specifications*

```
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 omezenií
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
```

Specifikace požadavků – úvod



Obrázek: Fáze specifikace software

Studie proveditelnosti

- Vstup: Soubor předběžných obchodních požadavků, základní popis fungování
- Neměla by dlouho trvat, neměla by být drahá $\Rightarrow$ bude se vůbec pokračovat?
- U nových projektů provádět vždy (také velké fáze běžících)
- Výstup: *Zpráva o proveditelnosti* odpovídající na otázky:
  - Splní(pomůže) systém hlavním cílům organizace?
  - Jsou požadavky uskutečnitelné s HW a SW možnostmi, rozpočtem a v daném termínu?
  - Může systém fungovat s již běžícími systémy, jaké úpravy?
- Odpovědi na otázky jsou kritické – často neznají cíle
- Sběr informací pro odpověď na otázky. Zdroje:
  - Manažeři lidí, kteří budou systém používat
  - SW inženýři, kteří podobný systém někdy navrhovali
  - Technologičtí experti
  - Koncoví uživatelé

Studie proveditelnosti

- Možné otázky:
  1. Co by organizace dělala, kdyby se systém neimplementoval?
  2. Jaké jsou současné problémy, které by měl systém pomoci řešit?
  3. Jak nový systém přímo napomůže splnit/zlepšit stav obchodních požadavků?
  4. Můžeme použít informace odjinud?
  5. Je vyžadováno nasazení nějaké nové(pro organizaci) technologie?
- Na základě zjištěných informací udělat sumarizaci
- Vydat doporučení, zda pokračovat dál
- Ve zprávě navrhnout změny požadavků – rozsah, rozpočet, rozvrh
  - Návrh základních abstraktních požadavků na systém

Zjišťováni požadavků

- V další fázi musím zjistit požadavky na systém a analyzovat je
- Zjišťovací fáze – rozhovory, spolupráce se zákazníkem, koncovými uživateli
- Hlavní cíl: Vyzískat co nejvíce (správných) informací k formulaci uživatelských a systémových požadavků
- Mnoho zúčastněných stran *(stakeholders)* – problémy v porozumění účastníkům
  - Nevědí, co přesně chtějí, schopni popsat pouze obecnou funkcionalitu
  - Nerealistické požadavky (neznají následky na cenu, dopady)
  - Vyjadřují se ve „svém jazyce“, používají své pojmy
  - Různí účastníci mají různé požadavky, různý styl vyjadřování (možný spor)
  - Mají politický zájem (nová funkcionalita někomu zvýší vliv)
  - Prostředí se může změnit během analýzy (noví účastníci, požadavky)

Zjišťování požadavků – fáze

1. *Objevování požadavků* – interakce s účastníky
   - Sběr doménových požadavků (a porozumění)
   - Dokumentování zaběhlých procesů
2. *Klasifikace a organizace* – třídění sesbíraných požadavků
   - Zjištěné požadavky mají různou formu – sjednocení
   - Seskupení podobných požadavků do souvisejících skupin
3. *Prioritizace a vyjednávání* – řešení sporů v požadavcích
   - Zjištění a stanovení, které požadavky „mají přednost“
4. *Dokumentace* – sumarizace předchozích fází
   - Podklad pro další fázi zjišťování

- Fáze v iterativním procesu (opakování), zpětná vazba

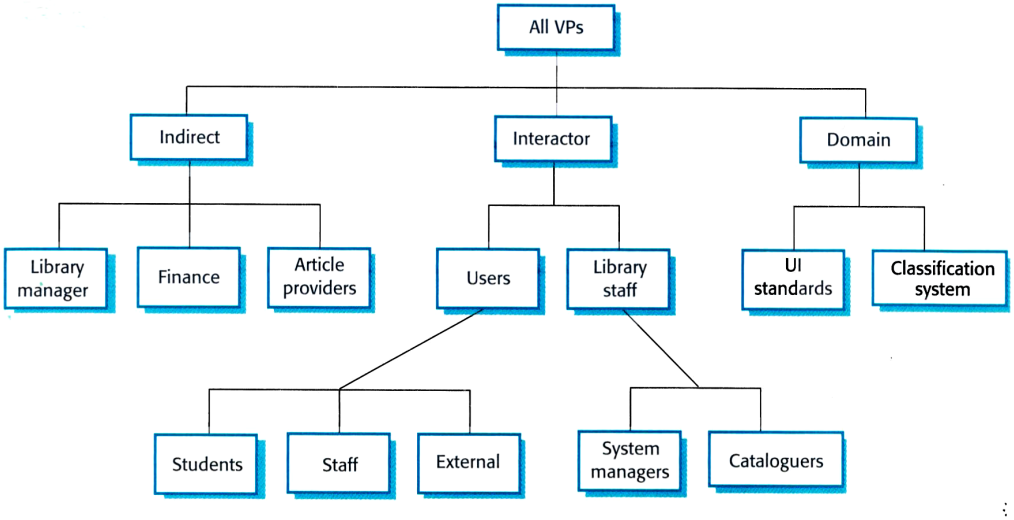
Objevování požadavků

- Při zjišťování bychom neměli zapomenout na nějaké účastníky systému
  - Systém by poté byl neúplný, nesloužil by dobře
- *Příklad: Mějme „operační“ systém pro bankomaty. Jací mohou být účastníci?*
  1. Zákazníci banky (kteří využívají bankomat pro výběr)
  2. Představitelé jiných bank (smlouva o užívání jiných bankomatů)
  3. Manažeři bankovních oddělení (budou dostávat informace)
  4. Správci bankovního systému (každodenní správa systému)
  5. Správci databáze (integrace systému s databází zákazníků)
  6. Bezpečnostní manažeři (zajišťují bezpečnost systému)
  7. Marketingové oddělení (reklamy na úvodní obrazovce)
  8. Správci HW a SW (údržba, aktualizace, instalace)
  9. Národní bankovní regulátor (kontrola legislativních pravidel)
- Každý bude mít úplně odlišný pohled na systém a odlišné požadavky
- Různé metody, jak požadavky *nejlépe* získat

Úhly pohledu

- Popisujeme systém z různých úhlů pohledu (*viewpoints*)
- Organizace procesu zjišťování a požadavků
- Klasifikace pohledů účastníků:
  1. *Přímý pohled* – přijdou se systémem přímo do styku (zákazník bankomatu, správce HW)
  2. *Nepřímý pohled* – nepoužívají přímo, ale ovlivňují (management, bezpečnost)
  3. *Doménový pohled* – doménové charakteristiky, podmínky na systém
- Zjištěné požadavky jsou rozděleny do těchto skupin
  - Přehlednost
  - Jasně definováno „odkud“ daný požadavek vzešel
  - Lépe se pak řeší prioritizace

Úhly pohledu – knihovní systém



Rozhovory

- Formální i neformální rozhovory s účastníky – nedílná součást zjišťování
- Uzavřené (pevné otázky) $\times$ otevřené diskuze
  - Výhody a nevýhody obojího
- Velmi dobré pro získání vhledu do celé problematiky
  - Uživatelé si rádi povídají o své práci
  - Jsou rádi, když máte zájem
- Pozor při zjišťování doménových požadavků
  - Časté použití žargonu, těžko se bez znalosti diskutuje, možná nedorozumění
  - Nějaká doménová znalost je účastníkům tak zažitá, že si ji ani neuvědomují (my ale neznáme)
- Neefektivní při odhalování organizačních vlivů (papír vs. realita)
- Pozor na „urvání se z řetězu s nápady“
- Vhodná spolupráce na prototypu

Scénáře

- Některou funkcionalitu je snadnější a srozumitelnější demonstrovat na konkrétním scénáři
  - Lépe představitelné než abstraktní popis (pro obě strany)
- Vhodné jako doplnění abstraktního popisu
- Demonstrace jednoho konkrétního, uceleného způsobu použití systému
- Scénář by měl obsahovat
  1. Popis počátečních podmínek a stavů pro scénář
  2. Popis normálního (bezchybného) průchodu scénářem
  3. Popis „co a jak se může pokazit“ a jak se s tím vyrovnat
  4. Popis aktivit, které mohou probíhat souběžně se scénářem
  5. Popis stavu systému po dokončení scénáře

Scénáře – příklad

Scénář 1 (Stažení článku z LIBSYS)

Předpoklad: Uživatel se přihlásil do LIBSYS a vyhledal si časopis se článkem.
Normální průběh: Uživatel zvolí článek ke stažení. Systém si vyžádá údaje o předplatiteli nebo nabídne platební metodu. Platba může být provedena kartou nebo poskytnutím čísla účtu organizace.
Uživatel vyplní formulář o autorských právech a odešle jej do LIBSYS.
Formulář je zkontrolován a pokud je schválen, je staženo PDF se článkem do interního LIBSYS úložiště. Uživatel je o dostupnosti informován. Následně se mu nabídne možnost zvolit tiskárnu.
Článek je vytištěn. Jestliže byl článek *pouze pro tisk* a byl úspěšně vytisknut, je smazán z úložiště.
Co se může pokazit: Uživatel správně nevyplní formulář o autorských právech. Měl by být uživateli vrácen k napravení chyb. Je-li i další verze nesprávná, vydání článku je odmítnuto.
Platba může být zamítnuta. Tisk článku je odmítnut.
Selže stahování článku. Systém bude pokus opakovat až do úspěchu či přerušení uživatelem.
Souběžné aktivity: Souběžné stahování více článků.
Stav systému po dokončení: Uživatel je přihlášen. Stažený článek byl smazán z úložiště, byl-li *jen pro tisk*.

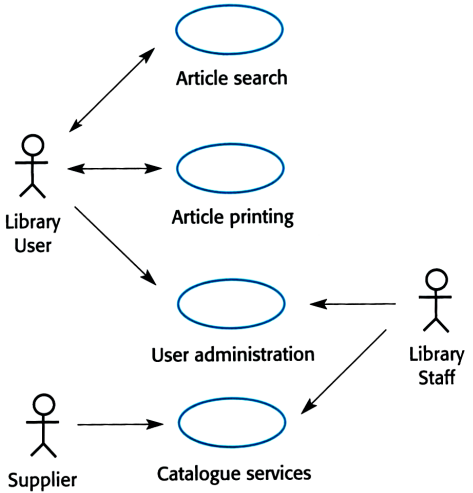
Případy užití

- Technika založená na scénářích, ale nejde do detailů – pohled „co vše se dá dělat“
  - Popis kontextu systému a funkčních požadavků
  - Vnější pohled na systém, zamýšlené funkce
- Případ užití(abstraktní) zahrnuje více scénářů (konkrétní)
- Využití základního diagramu z UML
- Uživatelé(a další systémy) – *aktéři* – v diagramu jako panáček s popisem role
- Možné interakce – *případy užití (use case)* – elipsa s názvem

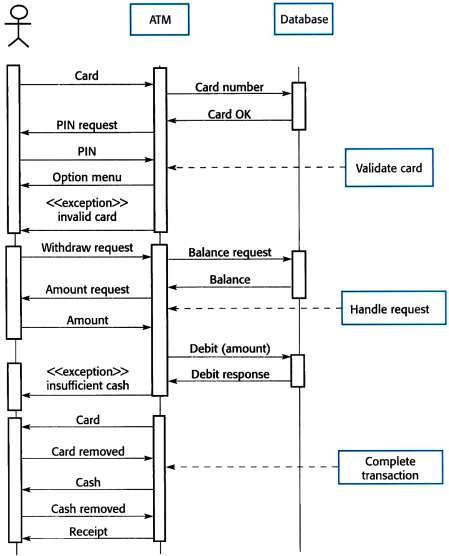
Případy užití – UML diagramy

- Často odrazový můstek pro podrobnější specifikaci
  - Máme „oddálený pohled“ na celkovou funkcionalitu
  - Můžeme rozpracovat do scénářů/systémových požadavků
- Zobrazení rolí a hierarchie uživatelů (generalizace)
- Rozšiřující funkcionalita `≪extends≫`
- Velmi efektivní pro získávání požadavku přímých účastníků
- Často spolu s interakčním diagramem

Případy užití – UML knihovny



Interakční diagram



Validace požadavků

- = Ověření, jestli požadavky definují opravdu to, co zákazník chce
- Měla by probíhat (přirozeně) průběžně během procesu
  - Nalezené chyby zanalyzovat a opravit
  - Čím dříve se chyba odhalím, tím lépe (následky, zbytečná práce)
- Měli bychom kontrolovat základní parametry:
  - Kontrola *oprávněnosti* požadavků – jsou požadavky opravdu potřeba nebo zbytečné?
  - Kontrola *konzistentnosti* – nejsou požadavky ve sporu (přímo, kruhem)
  - Kontrola *úplnosti* – jsou známé všechny okolnosti, pojmy?
  - Kontrola *realizovatelnosti* – zvládneme to technologicky? V termínu?
  - Kontrola *ověřitelnosti* – je možné ověřit splnění požadavku?
- Jak na to?
  - Revize/posudek – týmem, externí kontrolor
  - Prototypy – vyrobit jednoduchý prototyp, který osvětlí problematiku
  - Vytvoření testovacích případů – při vytváření můžeme odhalit nedostatky (ověřitelnost, úplnost)

Správa požadavků

- Požadavky se budou měnit
  - Jak během zjišťování
  - Tak během implementace a evoluce systému
- U velkých systémů může být velké množství
- *trvalé*(koncepční) $\times$ *nestálé* požadavky (konkrétní provedení)
- Monolitický dokument v jediné verzi není dobrý nápad
- Potřeba mít požadavky provázané (závislosti, prerekvizity, souvislosti)
- Potřebova udržovat historii verzí požadavků (opravy, důvody)
- Použití vhodného (existujícího) nástroje
  - Programy s databází $\Rightarrow$ vygenerování části DSP
  - Verzovací systémy (git) + pravidla evidence změn

Proces změny požadavků

- Požadavky bychom (ve specifikaci) neměli měnit jen tak svévolně
- Analýza problému a specifikace změny
  - identifikace problému nebo obdržení návrhu změny (od zákazníka)
  - zjištění, zda je to opravdu problém / validní návrh
  - $\Rightarrow$ podrobný návrh změny
- Analýza změny a její cena
  - určíme, jaké bude mít změna důsledky (závislosti)
  - jak bude potřeba změnit DSP, design a implementaci
  - odhadneme cenu a případnou časovou náročnost/termín
- Padne rozhodnutí o (ne) implementaci
- Implementace změny
  - Jak v kódu, tak v architektuře a (hlavně) v DSP
- Rozchod DSP a reality při urgentních změnách (nezapomínat, lavinovitý efekt)

Doporučená literatura

- Ian Sommerville. Software Engineering (10th Edition). Pearson, 2015. ISBN: 978-93-325-8269-9.
  - Chapter 4 – Requirements Engineering
- Joseph Ingeno. Software Architect's Handbook. Packt, 2018. ISBN: 978-1-78862-406-0
  - Chapter 3 – Requirements engineering and Requirements elicitation – p. 65-76
- Leszek A. Maciaszek. Requirements analysis and system design. Pearson Education. 2004. ISBN: 0-321-20464-6
  - Chapter 4 – Requirements Specifications