01 - Architektura informačních systémů

(význam, komponentová architektura a architektura pomocí služeb (SOA), modelem řízená architektura (MDA), další moderní přístupy k architektuře IS)

Význam - koncepční rámec řešení informačního systému. Dává budování IS nějaký směr a je vhodným komunikačním prostředkem mezi vedením podniku a projektanty IS.

Komponentová architektura a architektura pomocí služeb (SOA), modelem řízená architektura (MDA), další moderní přístupy k architektuře IS

- Komponentová architektura - IS je dekomponován do autonomních komponent (spustitelný program, knihovna, tabulka v databázi, …), které spolu komunikují pomocí rozhraní. Systém se tak dá přehledně sestavit z nakoupených komponent, které se mohou zvlášť testovat.
- Architektura pomocí služeb (SOA) - jednotlivé spolupracující komponenty systému jsou poskládány tak, aby odpovídaly službám reálného světa. Zákazník pak služby využívá pomocí rozhraní, které je odděleno od implementace. Výhodou je propojení s architekturou celého podniku, který se může zaměřit přímo na poskytované služby z katalogu služeb daného řešení.
- Modelem řízená architektura (MDA) - IS je většinou pomocí UML s OOP přístupem navržen jako model, kde aplikační logika je nezávislá na technologické platformě. Teprve poté se pomocí dalších vrstev modelů systém transformuje pro zvolenou platformu.
- Další moderní přístupy k architektuře IS - základem je klasická třívrstvá architektura - prezentační vrstva, aplikační vrstva, databázová vrstva. V některých případech (spíše webové aplikace) se tato architektura nahrazuje za MVC - model - view - controller, kde pohled je aktualizován přímo modelem na příkaz řadiče (trojúhelníková topologie).

02 - Vývoj informačního systému objektově

(iterativní a přírůstkový model, RUP a objektové metodiky. Objektová analýza; UML, struktura UML, prvky, vazby, diagramy, modely a pohledy, třídy a objekty, oblasti využití diagramů UML ve vizuálním modelování a jejich vztahy, objektové CASE nástroje)

Iterativní a přírůstkový model - základem je vodopádový model životního cyklu, který je však nevhodný pro větší projekty.

- Iterativní model - čtyři fáze (počátek, příprava, konstrukce a zavedení), každá z nich je z několika iterací s životním cyklem vodopád, každá z nich vede ke spustitelné verzi a obsahuje integraci a testování. Lépe se vyrovnává se změnou požadavků než vodopádový model.
- Přírůstkový model - inkrementální přístup rozdělí projekt na menší segmenty (snížení rizik, vyšší možnost změn v průběhu vývoje), které se provádí jako série malých vodopádů. Je vhodný pro kombinaci sekvenčních a iteračních metodik softwarového vývoje. Projekt je vytvářen po částech.

RUP a objektové metodiky

- RUP - Rational Unified Process, iterativní metodika vývoje softwaru (koncept) vhodná zejména pro rozsáhlé projekty. Po každé iteraci je k dispozici spustitelný kód, využívá se UML.
- Objektové metodiky - objektově orientovaný SW má výhodu ve znovupoužitelnosti komponent díky jejich zapouzdření a rozhraní a v komponentovém stylu vývoje systémů. Vzniklo několik objektových metodik návrhu IS (např. Booch, Jacobson, Fusion, OMT).

Objektová analýza - Díky objektovým metodikám se postupně vyvíjela ze strukturovaného návrhu. Je postavena na myšlence, že každý systém se skládá ze samostatných objektů - entit reality, které spolu komunikují pomocí zpráv a reagují na události. Výsledkem této analýzy bývá diagram případů užití, diagram tříd a interakční diagramy.

UML - UML je standartní jazyk pro vizualizaci, specifikaci, konstrukci a dokumentaci prvků projektu, ve kterém hraje významnou roli vývoj software.

Struktura UML, prvky, vazby, diagramy, modely a pohledy, třídy a objekty

- Prvky, vazby
    - Prvky - statické části modelu (třídy, rozhraní, use case, komponenta) a dynamické části modelu (interakce, stavy, aktivity). Dále pak vysvětlující prvky (popisy, anotace, poznámky).
    - Vazby - agregace, kompozice (rozdělení prvku na podčásti), asociace (propojení prvků), generalizace (dědičnost), realizace (vztah mezi třídou a rozhraním)

- Diagramy - grafická reprezentace obsahu modelu, zachycení prvků a jejich vztahů. Jednotlivé typy diagramů odpovídají pohledům na navrhovaný systém.
    - Use Case - zachycuje funkcionalitu systému z pohledu uživatele
    - Sekvenční - zachycuje interakci (zasílání zpráv) mezi objekty.
    - Spolupráce - zobrazuje interakci mezi objekty a jejich vzájemné vztahy.
    - Tříd - zobrazuje informace o třídách a jejich vazby, objekt je instancí třídy.
    - Stavový - modeluje životní cyklus jednoho objektu.
    - Aktivit - zobrazuje scénář případu užití.
    - Nasazení - ukazuje nasazení softwaru na fyzický hardware.
- Modely a pohledy, třídy a objekty
    - Modely - statický model (struktura - diagram tříd, komponent, nasazení) a dynamický model (chování - objektový, use case, sekvenční, spolupráce, stavový, aktivit).
    - Pohledy
        - Strukturální pohled - popisuje vrstvu mezi objekty a třídami, jejich asociace a možné komunikační kanály.
        - Pohled chování - popisuje, jak systémové komponenty interagují, charakterizuje reakce na vnější systémové operace.
        - Datový pohled - popisuje stavy systémových komponent (objekty) a jejich vazby.
        - Pohled rozhraní - je zaměřeno na zapouzdření systémových částí a jejich potenciální použití okolím systému.
    - Třídy a objekty - objekt je instance třídy. Pomocí asociace se v diagramu tříd vyjadřuje násobnost vztahu - kolik instance třídy A může být svázáno s instancí třídy B.

Oblasti využití diagramů UML ve vizuálním modelování a jejich vztahy

- Diagram use case -> dle objektů pak diagram tříd a dle scénářů sekvenční diagram a diagram aktivit (modelování dynamiky).
- Sekvenční diagram -> dle nových objektů pak diagram tříd, jinak rozšíření v diagramu spolupráce.
- Diagram tříd -> modelování dynamiky v sekvenčním diagramu, modelování metod objektů v diagramu aktivit, modelování životního cyklu jednoho objektu ve stavovém diagramu, tvorba autonomních komponent jako množina spolupracujících objektů v diagramu komponent.

Objektové CASE nástroje - například Enterprise Architect, Visual Paradigm. Nástroje pro modelování IT systémů pomocí diagramů, vytváření dokumentace a generování kódu z modelů či modelů z kódu.

03 - Řízení projektu vývoje IS

(význam modelování pro projekt, cíle a struktura projektu, úkoly, zdroje a náklady projektu, vlastnosti projektu, akceptace projektu, hlavní produkty a dokumenty projektu. Softwarové metriky a odhady. Řízení rizik v projektech)

Význam modelování pro projekt - napodobení (abstrakce) struktury a chování reálného systému. Usnadňuje řízení projektu ve všech jeho životních cyklech. Lépe se specifikují požadavky, náklady, rizika, zdroje, čas, ...

Cíle a struktura projektu

- Cíl projektu - základem dohody mezi zadavatelem a zhotovitelem, definuje výstupy projektu. Vytvořit nějaký nový předmět, službu nebo jejich kombinaci.
- Struktura projektu - projekt se dělí na subprojekty, ty pak na skupinu úkolů.

Úkoly, zdroje a náklady projektu

- Úkol - základní časová jednotka a činnost. Milník je úkol s nulovým nákladem, ale představuje důležitý projektový termín.
- Zdroje - zapezpečují činnost, jsou pod přímou kontrolou manažera. Jsou pracovní a materiální.
- Náklady - stanoveny rozpočtem, mají limit čerpání, jsou pevné nebo pohyblivé.

Vlastnosti projektu - je unikátní, má přesné cíle, dané termíny, danou cenu, omezené zdroje, začátek a konec.

Akceptace projektu - schválení výstupů projektu, předání výsledku.

Hlavní produkty a dokumenty projektu - smlouva projektu, základní dokument projektu, rozpočet projektu, zprávy o stavech a plnění projektu, protokol o převzetí a akceptaci, statistiky a znalostní báze.

Softwarové metriky a odhady - slouží k taktickým účelům, odhadům času, nákladů, monitorování projektu a vyhodnocení. Každá metrika má své vstupy, výstupy a výsledky.

- Metriky orientované na velikosti - počet řádek kódu, stránek dokumentace, chyb
- Metriky kvality software - správnost, udržovatelnost, integrita, užitečnost

Řízení rizik v projektech - riziko je přirozenou součástí projektu, má fyzickou a psychologickou stránku. Má negativní aspekt a cílem řízení rizik je jejich identifikace, minimalizace a vhodné řešení. Protiváhou rizika je rezerva. Příklady: cena (překročení rozpočtu), plánování (nedostatek zdrojů), technologie (nekompatibilita), ...

04 - Procesní řízení

(modelování podnikových procesů, reengineering podnikových procesů)

Proces je souhrn činností, které vyžadují vstupy a tvoří výstupy s hodnotou pro zákazníka.

Modelování podnikových procesů - modelování podnikových procesů slouží jako prvotní krok při zavádění reengeneeringu firemních procesů. Diagramy podnikových procesů jsou srozumitelné, jsou velice užitečné pro interakci mezi analytiky a pracovníky dané organizace a slouží jako podklady pro softwarové projekty. Diagramy firemních procesů pak dotváří úvodní studii pro softwarové projekty. Díky těmto diagramům je mizivá šance, že dojde k opomenutí základních požadavků na SW projekt a ovlivňuje jeho tvorbu. Takto vzniklé diagramy slouží pro vyhodnocení efektivnosti jednotlivých projektů.

- Můžeme využít některé diagramy UML, ale spíše BPMN (Business Process Modeling Notation) - diagram procesu obsahuje cíl, vstup, výstup, podpůrné objekty a řídící objekty.

Reengineering podnikových procesů - procesy je nutné neustále zlepšovat, aby odpovídaly zvyšujícím se nárokům zákazníků, kteří by jinak hledali u konkurence. Můžeme je zlepšovat průběžně v cyklech přírůstkově na základě porozumění a měření stávajícího procesu nebo reengineeringu podnikových procesů.

- Reengineering předpokládá že současný proces je zcela nevyhovující a je nutné jej od základu změnit (vady, vyjímky, složitost, nutná častá kontrola). Dochází k radikální rekonstrukci za účelem maximálního zdokonalení z hlediska měření výkonnosti. Probíhá v několika fázích - stanovení rozsahu a cíle projektu, analýza potřeb a možností nových technologií, definice nové soustavy procesů, plán zavedení a přechod na nový proces.

05 - Softwarové inženýrství

(proces vývoje softwaru, softwarový projekt, plánování projektu, jeho řízení a rizika)

Softwarové inženýrství je metodologická disciplína pro produkci programů.

Proces vývoje softwaru - koncept, plán, realizace, předání

Softwarový projekt - činnost omezena náklady a časem, jejímž cílem je dosažení souboru definovaných přínosů dle patřičných standardů a požadavků kvality.

Plánování projektu, jeho řízení a rizika

- Plánování projektu
    - Plánování na základě tří znalostí
        - Kde nyní jsme
        - Kam se chceme dostat
        - Jak se tam dostaneme
    - Trojimperativ projektového managementu
        - Čas - který je limitní pro plánování sledu jednotlivých dílčích aktivit projektu.
        - Zdroje - které jsou projektu přiděleny a které budou užívány a čerpány
        - Náklady - které jsou finančním projevem užití zdrojů v časovém rozložení.
    - Formulace cílů projektu pomocí SMART
        - S Specific - cíle mají být specifický a konkrétní.
        - M Measurable - mají být opatřeny měřitelnými parametry, podle nichž lze rozpoznat, zda bylo cíle dosaženo.
        - A Assignable - cíle mají být přidělitelné jedinému subjektu s odpovědností a autoritou k výkonu rozhodnutí.
        - R Realistic - cíle mají být dosažitelné s použitím disponibilních zdrojů a realistické.
        - T Timebound - cíle mají být časové ohraničené.
- Řízení projektu - řízení projektů je soubor modelů, metod, postupů, nástrojů a technik pro plánování a řízení realizace složitých projektů.
- Rizika projektu - riziko je přirozenou součástí projektu, je třeba ho rozpoznat, snížit na únosnou míru a zbývající riziko zvládnout. Rezervu lze chápat jako protiváhu rizika, pokud lze do projekt vložit velkou rezervu, riziko se sníží. Každá dimenze trojimperativu by měla mít svoji rezervu, ale nejčastěji se týká času a nákladů. Předpokladem je jasná dokumentace všech prvků, které mohou ovlivnit rizika na všech úrovních podrobností.

06 - Metodologie softwarového inženýrství (SI)

(obecné principy SI, klasifikace a charakteristiky metodik SI, agilní přístupy, RAD)

Obecné principy SI

- Princip modelování - informační systém je modelem reálného systému
- Princip iterace - analýza a návrh se provádějí v postupných krocích, které se zpřešňují
- Princip strukturování - systém se rozdělí na menší celky a postupně se řeší
- Princip životního cyklu - zachycuje celý proces vývoje
- Princip automatizace - využívá se počítačové podpory po celou dobu životního cyklu

Klasifikace a charakteristiky metodik SI, agilní přístupy

- Rigorózní (heavy-weight) metodiky - podrobně a přesně popisují jednotlivé procesy vývoje software, definují posloupnost činností, obsahují objemné dokumentace. Formulace požadavků je fixní, variabilní jsou zdroje a čas.
    - Vodopádový model životního cyklu - lineární resp. sekvenční typ frameworku (malá pružnost). Projekt je rozdělen na fáze jdoucí postupně za sebou (analýza požadavků, návrh, implementace, testování (validace), integrace a údržba), přičemž některé se mohou překrývat. Autor model označil jako příklad chybného, nefungujícího modelu. Důraz je kladen na plánování, časové rozvrhy, termíny, rozpočty a realizace celého systému najednou. Přísná kontrola je udržována po celou dobu životnosti projektu prostřednictvím rozsáhlých písemných dokumentů, formálních revizí a schvalování uživatelem. Hlavní nevýhoda: nedá se vracet.
    - Spirálový model životního cyklu - kombinace sekvenčního a iterativního typu frameworku (prototypování a designování). Rozdělí projekt na menší segmenty (snížení rizik, vyšší možnost změn v průběhu vývoje). V průběhu vývoje je také možné vyhodnocovat rizika a zvažovat další pokračování projektu v průběhu životního cyklu. Každý cyklus spirály spouští stejný sled kroků pro každou část produktu a pro každou úroveň elaborace (od konceptuálních dokumentů až po programování jednotlivých programů). Během každého cyklu spirály jsou spouštěny čtyři základní fáze (kvadranty): (Analýza – stanovení cílů, alternativ a rozsahu iterace; Vyhodnocení – vyhodnocení alternativ, identifikace a řešení rizik; Vývoj – vývoj produktu a kontrola očekávaných výsledků; Plánování – plán pro příští iteraci). V počátku každého cyklu se identifikují zainteresované subjekty a jejich podmínky kladené na úspěch iterace, na konci každého cyklu se provádí revize a předání.
    - Prototypový přístup - Vývoj pomocí vytváření prototypů, dochází k vývoji neúplných verzí software, tzv. prototypů. Není samostatným a kompletním přístupem metodiky vývoje, ale spíše přístup k jednotlivým částem větších tradičních metodik vývoje software (přírůstková metoda, spirála, nebo RAD).

Rozdělí projekt na menší segmenty (snížení rizik, vyšší možnost změn v průběhu vývoje). Uživatel je zapojen v celém procesu vývoje, což zvyšuje pravděpodobnost přijetí konečné implementace uživatelem. Malé ukázky systému jsou vyvíjeny iterativním procesem, dokud se prototyp nevyvine tak, že splňuje požadavky uživatele. Většina prototypů je sice vyvíjena s tím, že budou vyřazeny, ale v některých případech je možné pokročit od prototypu k funkčnímu systému.
    - Rational Unified Process (RUP) - iterativní metodika vývoje softwaru (koncept) vhodná zejména pro rozsáhlé projekty. Po každé iteraci je k dispozici spustitelný kód, využívá se UML.
- Agilní metodiky - zaměřují se na velmi rychle vytvořené řešení a pružně se přizpůsobovat měnícím se požadavkům. Upřednostňují rychlý vývoj a následné úpravy dle zpětné vazby od koncového uživatele. Požadavky na funkcionalitu se přizpůsobují v průběhu vývoje, čas a zdroje jsou fixní. Velmi krátké iterace, časté změny v kódu, komunikace se zákazníkem, menší dokumentace.
    - Extrémní programování
    - SCRUM
    - Test-driven development

RAD - Rapid Application Development, rigorózní metodika s náběhem do agilní metodiky. Iterativní vývoj prototypů. Snaha o velmi rychlý vývoj vysoce kvalitních systémů při relativně nízkých investičních nákladech. Umožňuje změny během procesu vývoje a snižuje rizika rozdělením projektu na menší segmenty. Aktivně zapojuje uživatele a automatizované vývojové nástroje (CASE). Důraz je kladen na naplňování business potřeb.

07 - Analýza požadavků na software

(požadavkové inženýrství, specifikace požadavků, zásady tvorby specifikace, modelování požadavků)

Požadavkové inženýrství - před objektově orientovanou analýzou a objektově orientovaným designem je potřeba vytvořit přehled o tom, co má systém dělat a jaký je smysl a specifikace požadavků. Požadavkové inženýrství je stanovení služeb, které by měl vyvíjený systém poskytovat, včetně různých omezení. Základem jsou informace o podniku zadavatele.

Specifikace požadavků - úplný popis chování systému. Soubor use casů (funkční požadavky) a omezení na design, provedení, rychlost, cenu, výkon, bezpečnost, … (nefunkční požadavky).

Zásady tvorby specifikace - požadavky mohou být popisovány formálně (přesný textový popis), neformálně (často nejednoznačný textový popis), UML Use case (popisuje použitá daného systému) nebo User Stories (používá běžný jazyk, u agilních metodik).

Modelování požadavků - v praxi se můžeme setkat s prototypováním aplikací - vytvořením jednoduchého, nefunkčního prototypu sloužícího jako vzor výsledného produktu.

08 - Návrh software

(architektura softwarových systémů, modelování softwarových systémů, návrhové vzory)

Architektura softwarových systémů - základní organizace systému dána jeho komponentami, vzájemnými vztahy těchto komponent a jejich vztahy k okolnímu prostředí a principy řídícími její návrh a evoluci. Tyto komponenty jsou uspořádány tak, aby umožnili provádění určité funkce či funkcí.

Modelování softwarových systémů - modely životního cyklu, které obsahují vlastní metodiku, jak zajistit dostatečnou kvalit softwarového produktu. UML modelovani.

Návrhové vzory - popis řešení problému návrhu nebo šablona, která může být použita v různých situacích.

- Vzory o tvorbě objektů
    - Singleton (jedináček) - potřebujeme-li, aby třída měla maximálně jednu instanci.
- Vzory o struktuře programu
    - Adaptér - potřebujeme-li, aby spolu pracovaly dvě třídy, které nemají kompatibilní rozhraní. Adaptér převádí rozhraní jedné třídy na rozhraní druhé třídy.
- Vzory o chování
    - Iterátor - zajišťuje možnost procházení prvků bez znalosti jejich implementace.

09 - Tvorba software

(paradigmata tvorby software, vývojová prostředí, možnosti použití CASE nástrojů, automatizace tvorby SW, verzování, modularizace a API)

Paradigmata tvorby software

- Imperativní - někdy také zvané procedurální, založeno na myšlence definice “jak se to má udělat” pomocí posloupnosti příkazů. Vhodný zejména pro malé projekty. Mohou být strukturované, které se někdy dělí dále na modulární a nemodulární, (C, Pascal) a nestrukturované (Fortran). Nejdůležitějšími příkazy jsou přiřazování, cykly a větvení.
- Deklarativní - založeno na myšlence definice “co se má udělat”, specifikují cíl a algoritmizace je ponechána interpretu jazyka. Neobsahují cykly, vše je řešeno pomocí rekurze. V programech nebývá důležité pořadí jednotlivých pravidel, protože kód nebývá zpracováván lineárně. Dělí se na funkcionální jazyky (Haskell nebo hybridní Lisp) založené na lambda-kalkulu (Church) a logické jazyky (Prolog) založené na predikátové logice.
    - Funkcionální - výpočtem je posloupnost vzájemně ekvivalentních výrazů, které se postupně zjednodušují až na nezjednodušitelnou normální formu. Výhodou je, že funkce nemají vedlejší účinky, které mění stav programu. Používají se hlavně v akademické sféře.
    - Logické - program je zadán v podobě množiny faktů a pravidel (Hornovy klauzule - predikátová logika prvního řádu), tedy tvrzení a systém se jej snaží dokázat pomocí unifikace, rekurze a backtrackingu.
- Objektově orientované - vhodné pro rozsáhlé projekty. Základním prvkem jsou objekty, které modelují objekty reálného světa a mají stav, který mohou měnit. Objekt v sobě zapouzdřuje data a metody, průběh výpočtu je určen posíláním zpráv mezi objekty, které říkají jak má objekt změnit svůj stav. Je založen na sigma kalkulu a nejzákladnějším přístupem jsou třídní OO jazyky, kde objekt vzniká jako instance třídy (Java). Méně rozšířené jsou prototypové OO jazyky, kde objekt vzniká dědičností z prototypu (Javascript). Některé jazyky jsou jen hybridním rozšířením imperativního o objekty (C++).
- Aspektově orientované - poměrně nové paradigma, které vylepšuje objektově orientované paradigma o možnost přidání aspektu, řešení průřezového problému (například logování), které by se jinak muselo implementovat na mnoha místech v kódu (částečně Enterprise Java Beans).

Vývojová prostředí - (IDE) software usnaďnující práci programátorů, většinou zaměřené na jeden programovací jazyk, obsahuje kompilátor, linker, debugger, často GUI (NetBeans, Eclipse, MS Visual Studio, CodeBlocks, Atom).

Možnosti použití CASE nástrojů - nástroje podporující vývoje sw aplikací, CASE - Computer Aided Software Engineering (např. Enterprise Architect). Umožňují generování zdrojového kódu a dokumentace z modelů. Původní myšlenka byla, že nebude potřeba programátorů a kód bude generován pomocí CASE nástrojů, to se však nepovedlo. Přesto zajišťují vyšší produktivitu práce, nižší chybovost, snažší údržbu a vývoj, kvalitnější dokumentaci a modelování chování budoucí aplikace.

Automatizace tvorby SW - automatizovat se dá zejména tvorba dokumentace a testů, v omezené míře také základy programů (třídy z diagramu tříd, tabulky databáze z ER diagramu).

Verzování - uchovávání historie všech provedených změn v kódu, většinou pomocí rozdílů mezi jednotlivými revizemi (Subversion (SVN), GIT, Dropbox).

Modularizace a API

- Modularizace - rozdělení logických částí softwaru do modulů. Většinou v imperativním a objektově orientovaném programování.
- API - (Application Programming Interface) - rozhraní pro programování aplikací. Sbírka funkcí či tříd v knihovně, které může programátor využívat a způsob jejich vykonání pak záleží na konkrétním počítači a HW (OpenGL, Posix) nebo rozhraní pro komunikaci s nějakou webovou aplikací, většinou pomocí XML nebo JSON (WebAPI).

10 - Dokumentace softwarového projektu

(dokumentace a její význam, typy dokumentace, dokumentace dle fází životního cyklu, programová dokumentace, automatizace tvorby dokumentace)

Dokumentace a její význam - textový dokument, vyžadovaný standard z důvodu rychle vzrůstající složitosti informačních systémů a jejich vzájemné provázanosti ve velkcýh společnostech.

Typy dokumentace

- Dokumentace požadavků - popis, co software vykonává, identifikuje atributy, schopnosti a vlastnosti systému. Tvoří základ pro to, co bylo nebo bude realizováno.
- Dokumentace architektury / designu - přehled softwaru. Zahrnuje vztahy k prostředí a stavebním základům, které budou použity v návrhu sw komponent.
- Technická dokumentace - dokumentace kódu, algoritmy, popis rozhraní a API. Vzniká současně při tvorbě softwaru. Dá se částečně automatizovat.
- Uživatelská dokumentace - manuály pro koncového uživatele, systémové administrátory atd. Většinou jednodušší, bez příliš složitých technických detailů (dle cílové skupiny).
- Marketingová dokumentace - jak prodávat produkt, analýza tržní poptávky, propagační materiály, stručné a výstižné představení aplikace.

Dokumentace dle fází životního cyklu

- Specifikace požadavků - specifikační dokument vytvořený na základě vzájemné domluvy řešitele a uživatele (zadavatele). Mívá právní platnost a je součástí smlouvy. Obsahuje zejména požadovaný výsledek a podmínky dodání. Vzniká z neformální specifikace, která se transformuje na formální specifikaci obsahující funkční (funkce) a nefunční (design, provedení) požadavky. Návrh řešení požadavků a odhady nákladů a času pak obsahuje úvodní studie na základě které se schvaluje projekt.
- Analýza - výsledkem je model systému (diagramy a popisy), vytváří se hrubé návrhy funkcí a datových struktur (konceptuální modely), ze kterých se pak vytvoří detailní a úplné datové a funkční modely.
- Návrh - vezmeme v úvahu jazyk a vývojové prostředí (HW, SW). Vstupem je funkční specifikace systému, popisy funkcí, uživatelského rozhraní a návrh databáze. Výstupem je podrobný plán kódování a testování. Také obsahuje popis rozhraní včetně uživatelských obrazovek.
- Implementace - programy jsou podrobně dokumentovány, je vytvořena provozní dokumentace IS a uživatelská příručka. Vzniká programová dokumentace.
- Testování - plány testování vznikají už ve fázi analýzy a návrhu, měly by být součástí smluvy. Zde se doplňuje dokumetnace chyb.

Programová dokumentace - synonymum k technické dokumentaci.

Automatizace tvorby dokumentace - Technická dokumentace - Doxygen (univerzální, z komentářů zdrojového kódu, výstup v podobě PDF, LaTeX, HTML) nebo Javadoc (Java, HTML s CSS), …

11 - Metriky softwarových systémů

(klasifikace metrik, využití metrik a jejich zjišťování, kvalita softwaru)

Metriky produktu, procesu a projektu jsou kvalitativní charakteristiky programů, procesu jejich tvorby a projektu. Důvodem je kontrola kvality produktu z důvodu plánování či zdokonalení.

Klasifikace metrik

- Metriky orientované na velikost - počet chyb na tisíc řádků kódu, počet defektů na tisíc řádků kódu, cena řádku kódu, počet stran dokumentace na tisíc řádků kódu, počet chyb za člověkoměsíc, počet řádků kódu za člověkoměsíc, cena stránky dokumentace
- Funkčně orientované metriky - počet logicky různých vstupů, počet výstupů, počet dotazů, počet vnitřních logických souborů, počet souborů na rozhraních

Využití metrik a jejich zjišťování

- Přímá měření - počet řádek kódu, rychlost výpočtu, velikost paměti, počet chyb za určitou dobu
- Nepřímá měření - funkčnost, kvalita, složitost, pracnost, spolehlivost, schopnost údržby

Kvalita softwaru - je shoda s explicitně stanovenými požadavky na funkci a chování, explicitně dokumentovanými vývojovými standardy a implicitními charakteristikami, které se očekávají od každého profesionálně vyvinutého softwaru.

12 - Validace a verifikace softwaru

(metody a techniky testování, akceptační testy, funkční a zátěžové testování, refaktoring)

Verifikace - odpovídá software našemu návrhu?
Validace - plní software to, co se od něj očekává?
Chyby - každý sw má chyby, testováním můžeme dokázat, že sw chyby obsahuje, nikolik, že je bez chyb (nedokážeme simulovat nekonečné množství vstupních variací).

- Chyba ve specifikaci - nedostatečná, nadbytečná, rozporná specifikace
- Chyba software - syntaxe, sémantika, chybné výpočty, nesoulad se specifikací, bezpečnostní chyba

Metody a techniky testování

- Manuální vs automatické - automatické například skriptem, manuální buď formální (dle scénáře) nebo neformální (tester zkouší).
- Statické vs dynamické - statické je testování zdrojového souboru, dynamické je testování spušťeného programu.
- BlackBox vs Whitebox - BlackBox je testování na základě vstupu / výstupu, WhiteBox je testování vnitřních komponent a architektury.
- Unit testing - jednotkové testy pro velmi malé části vnitřního softwaru (funkce, komponenta).

Akceptační testy - test provedený zákazníkem, ověřuje, že sw splňuje specifikace.

Funkční a zátěžové testování

- Funkční testování - BlackBox testování, srovnáváme výstup se specifikací software.
- Zátěžové testování - testujeme maximální stanovenou zátěž SW/HW.

Refaktoring - změna architektury programu, předělání existujících rozhraní, jejich doplnění či zjednodušení. Příklad: snížení objemu zdrojového kódu, odstranění redundance, odstranění magických čísel, zabstraktnění kvůli znovupoužitelnosti, zavedení parametrizace kvůli flexibilitě atd.