1. Základní schéma životního cyklu software. Pracnost jednotlivých etap. Techniky specifikace požadavků. Varianty životního cyklu. SW prototypy. Strukturovaný vývoj. SW metriky a jejich využití. Techniky odhadu pracnosti a doby řešení. Funkční body. COCOMO. Kvalita SW, techniky zajitění kvality, ISO 9000.

Základní schéma životního cyklu software

Software je tvořen celkovým souhrnem počítačových programů, procedur, pravidel a průvodní dokumentace a dat, která náleží k provozu počítačového systému.

- Je vyvíjen a řešen inženýrskými pracemi, není vyráběn v klasickém slova smyslu.
- Fyzicky se neopotřebuje
- SW produkty: 1.genericky SW: su vytvarane pre lubov. zakaznika na trhu, specifikaciu vytvara obchodne oddelenie 2.zakazkovy SW: objednane urcitym zakaznikom, specifikacia tvori dolezitu sucast kontraktu medzi zakaznikom a dodavatelom

Jeho životní cyklus začíná nápadem na to začít vymýšlet nějaký software a končí v okamžiku kdy jej poslední člověk přestane používat.

Mezitím lze jeho životní cyklus popsat zhruba takto:

1. **Specifikacia problemu:** vytvorenie dostatocne presnej specifikacie problemu, navrh sposobu ovladania programu, kontrola specifikacie(odhalenie nejednoznacnosti, neuplnosti v specif., overenie toho ci problem popisany specifikaciou, skutocne povodne zamyslany problem)
2. **Analyza:** vytvorenie logického modelu riesenoho systemu a jeho zaznam pomocou grafickych technik
3. **Navrh:** dekompozicia riesenia na programatorsky zvladnutelne casti, dva sposoby: navrh zhora nadol: problem je rozkladany na mensie podprogramy tak dlho az je specifikacia ciastkovych podproblemom dostatocne mala pre zahajenie programovania . Vyuziva sa pri tvorbe noveho SW. Navrh zdola nahor: vychadza z postupneho budovania zloziek produktu od jednoduchsich casti k zlozitejsim. Vyuziva sa pri modifikacii exist.SW.
4. **Implementacia:** navrh algoritmu s implementacia jednotlivych casti specifikacie a ich programovanie v progr. jazyku.
5. **Testovanie:** validacia produktu proti specifikacii
6. **Prevadzka a udrzba :** udrzba a vyvoj verzii, oprava chyb

Dobře řešený SW: udržovatelnost, spolehlivost, efektivita, použitelnost.

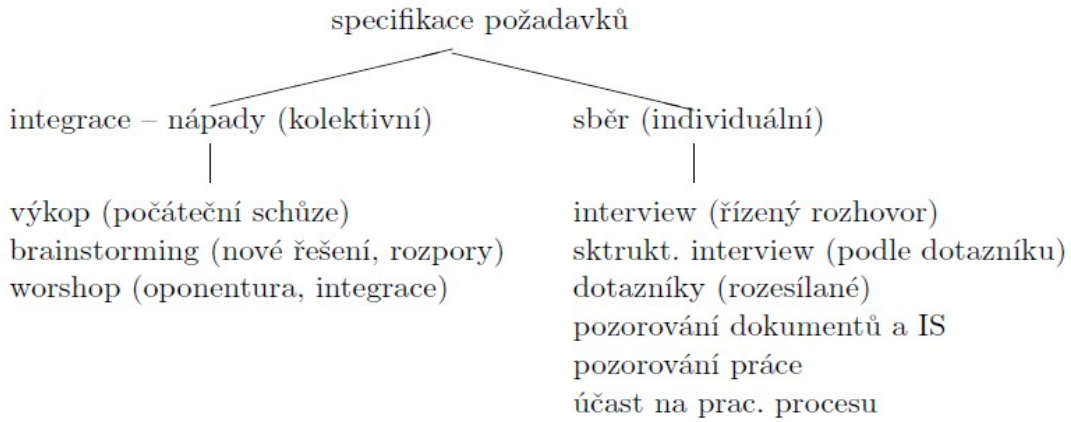
Pracnost jednotlivých etap.

- Obecně nejpracnější činností je vytváření dokumentace a odstraňování defektů v programech.
- U velkých systémů zabere 90% času odstraňování problémů.
- Obecně platí, že ve zmíněném životním cyklu je velmi obtížné vracet se ob více než jeden krok zpět. Tj. opravovat chybu ve specifikaci až když je projekt předaný je velmi obtížné.

- Az 85% nakladov ide na opravy chyb urobenych v etape vizia a specifikacie a nie na programatroske pochybenie .
- cena opravy chyby sa nasobi cca 3-5 krat na kazdu etapu, ktorou prejde neodhalena napr. ak hu neodhalime pri specifikacii je to 1 a aksa odhali pri navrhu uz je to 5 atd.

Techniky specifikace požadavků.

Špecifikácia požiadaviek je zároveň základom, ale aj úzkym miestom každého systému. Môžeme si ho predstaviť ako dokument, ktorý združuje všetky užívateľské požiadavky na systém prijatý od užívateľa, v štandardizovanej a štrukturovanej forme. Ide vlastne o vyjadrenie toho, čo by mal systém robiť a nie, ako by to mal systém robiť. Ideální specifikace požadavků by měla být naprosto pevná, přesná, úplná a bezesporná.Dokument Specifikace požadavků je závazným podkladem pro návrh a realizaci systému.



- **Interview**:hlavny prostriedok ziskavania poziadavok, dobre pripraveny pohovor o tom, co uzivatel robi, co by mohol IS zlepsitci priniest,Varianta porady, kde se jedni převážně ptají a druzí odpovídají
- **Strukturované interview:** interview, kde sa postupne odpoveda na otazky podladopredu pripraveneho dotaznika
- **Dotaznik:** podobné strukt. interview – pouze dostane na stul a nikdo mu s tím neradí, výhoda: pro rychlé zjištení relativne jednoduchých informací + levné
- **Studium dokumentu** (a existujícího IS): výhoda: vzor rešení (napr. obrazovek) + levne, nevýhoda: nebezpecí opakování chyb
- **Úcast/sledování prac. procesu**: výhoda: nezprostredkovaná informace, nevýhoda: výberové efekty (nezachytí se cinnosti vyskytované zrídka), velmi drahé, ruší to
- **Brainstorming:** Neformální porada s cílem najít nová řešení, vize a myšlenk, Okamžité nápady se hned zapisují na flipcharty a obvykle neformálně hodnotí, i bláznivé nápady se nezatracují a zapisují, vyhodnocení a koordinace nápadů již nebývá součástí porady
- **Workshop:** Pro hodnocení a kontrolu průběhu prací, získání přehledu o stavu prací
- **Oponentury:** nejefektivnjší detekce anomáli, snížení rizika neúspchu ale pracné,detekuji ale nemám opravovat,vnutorne: **Revizia:**Verifikace většího celku formou blízkou oponentuře v běžném smyslu. Používá se i u feasibility study (studie uskutečnitelnosti) **Inspekc**e: Přísně formalizované oponentní řízení pro menší dokumenty s řadou činností a rolí. vonkajsie:**Audit:**Varianta porady, kdy se má zjistit, zda se řešení (ekonomicky) neodchyluje od plánu a zda je naděje na dosažení cílů co do obsahu i termínů

Varianty životního cyklu.

Model vodopád

Patri medzi velmi rozsireny a jeden z najstarsich modelov vyvoja SW. Po uplnom dokonceni jednej etapy je vysledok tejto etapy odovzdany ako vstup pre etapu dalsiu. K zakoncenej etape nieje potrebene sa vracat. Tento model je zlozeny s postupnosti jednoznacne vymedzenych etap, ktore na seba nadvazuju a vzajomne sa neprelinaju. Problemy: chyby, ke ktorým je potreba se vracet – zvyšují se náklady a zákazník vidí svůj software až na konci procesu, což je problém v okamžiku, kdy na začátku procesu vlastně ani neví co chce.Tento model se hodí pro oblasti, které řešitelský tým dobře ovládá.

Model výzkumník

Tento model je vhodný řekněme pro programování věcí, které řešitelé neznají. Tento systém práce se obtížně řídí a plánuje, má problémy s dokumentací (neexistující či neplatná) a zpravidla do vývoje vidí jen jeden výzkumník nebo výzkumný tým a jejich rozpad či odchod pracovníka znamená ukončení projektu (nenahraditelnost řešitelů). Je to experimentování, u kterého často netušíme, jak dopadne. Pocas vyvoja sa riesitelia pri ziskavani poznatkov a skusenosti casto vracaju k uz prezitym etapam.

Skládá z operací: `Vytvoř systém → Implementuj systém → Používej systém. → Pokud vyhovuje, předej systém. Pokud ne, vrať se k nové implementaci.`

Prototypování

Tato metoda se snaží překonat problémy spočívající v nepřesně formulovaných případně měnících se požadavcích na systém. Spočívá ve vytváření tzv. *protopypů*, což jsou částečné implementace výsledného produktu se všemi rozhraními, které pak potenciální uživatelé testují, slouží k získávání poznatků. Prototypy se pak případně vylepšují, použijí nebo zahodí. Každý prototyp se vyhodnotí a v závislosti na něm se upravuje obrysová specifikace. Po posledním prototypu máme k dispozici finální specifikaci a vytváříme finální produkt.Prototypování se obvykle používá jen u menších systémů a zpravidla je nutné stanovit hranici pro vytváření prototypů, aby se nevytvářely donekonečna.

Spirálový model

Vytvořený hlavně za účelem minimalizace rizik při postupu pomocí vodopádu. Jednotlivé kroky se ve spirále opakují se stále vyšším a vyšším stupněm zvládnutí problematiky. Spirála většinou začíná analýzou rizik, dále se rozvíjí prototyp a zakončuje se kontrolou výsledků a testováním. Kazda faza vodopadu je tu riesena zhodnym postupom, kt sa sklada z ciatkovych krokov.vhodné pro IS, kde je znaná míra nejistoty ve stanovení požadavku a pro vývoj od pocátku, vhodné pro velké systémy

Iteračný model

Iterativně = „předělávat“.Celý projekt se vyvíjí v několika iteracích, Iterace směřují k postupnému vylepšení, zpřesnění, dodělání nebo opravení části systému. Každá iterace obsahuje analýzu, návrh, testování apod. (tj. miniaturní vodopád),ale s různou intenzitou, např.:

- V první iterace provést celkovou analýzu požadavků a obrysový plán vývoje,více rozpracovat jádro systému, implementovat základní testovací třídy
- V druhé iteraci podrobně rozpracovat důležité části systému, rozmodelovat je a částečně implementovat
- Ve třetí iteraci podrobně rozpracovat méně podstatné části systému, doimplementovat věci z předchozí iterace

Vývoj typu vodopád je tedy iterativní vývoj s jedinou iterací

Inkrementální model

Inkrementálně = „přidávat k“. Uplatňuje se zejména u větších projektů a/nebo v agilním vývoji, Jednotlivé části systému (přírůstky, inkrementy) vytváříme „nezávisle“ na zbytku a pak integrujeme

Vývoj jednotlivých přírůstků může probíhat iterativně, vodopádem, XP, …Nejčastěji se používá iterativní vývoj přírůstků. Vývoj typu vodopád je tedy inkrementální vývoj s jediným přírůstkem představujícím celý systém.

Strukturovaný vývoj.

Strukturované modelování:Oddělené funkční a datové modely,Postupné zpřesňování modelů,Snaha o zachování konzistence: uvnitř modelu a vzájemně mezi jednotlivými modely, Uspořádáním funkcí hierarchicky (funkční model) a dat (datový model)Usnadňuje orientaci ve funkčním a datovém modelu. Stále velká složitost vztahů zejména mezi funkčním a datovým modelem.

Strukturovaný vývoj je založený na strukturované analýze, což je druhá část vývoje projektu. První část vývoje je soubor uživatelských požadavků na systém. Původně s ní přišel pan DeMarco (1978) Strukturovaná analýza a specifikace systému (SASS) Analýza pomocí 4 modelů: studie stávajícího fyzického modelu, odvození logického ekvivalentu stávajícího modelu, odvození nového logického modelu, odvození nového fyzického modelu.Základním cílem tejto metody ale vsetkych strukturovanych je:

- Rozdělení problémů na menší, dobře definovatelné aktivity a určení posloupnosti a interakce těchto aktivit.
- Používají diagramy a další techniky k vytvoření specifikace, které rozumí uživatelé i návrháři systému.
- Rozdělení projektu na jednotlivé etapy a dílčí kroky umožňuje lepší odhad času, kontrolu výsledků a včasné řídící zásahy.

Pro modelování systému používá metodika tyto nástroje:

- DFD
- Datový slovník
- Strukturovanou angličtinu
- Rozhodovací tabulky procesů
- Rozhodovací stromy

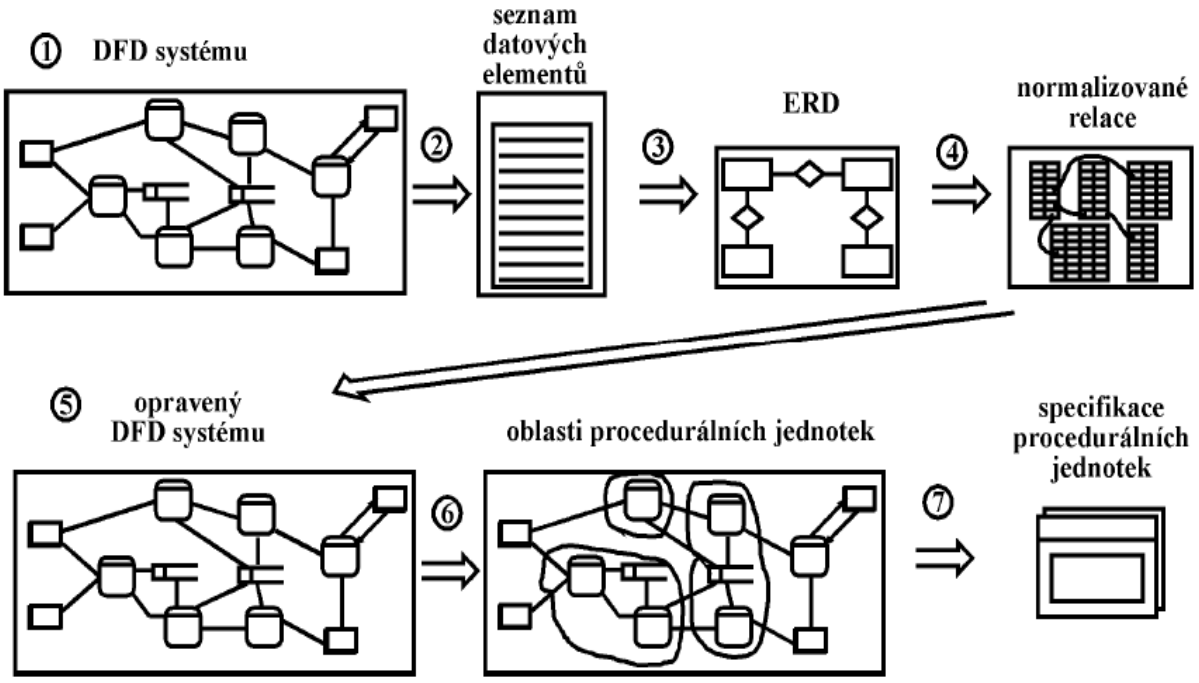
DFD(Data Flow Diagram): nástrojů pro modelování funkcí systémů, ale aj modelovací nástroj, který umožňuje zobrazit systém jako síť procesů, které plní určené funkce a předávají si mezi sebou data. Tímto způsobem jde modelovat nejen toky dat, ale i toky fyzických předmětů v systému. Diagram obsahuje čtyři základní typy komponent:

- Terminátory – reprezentují externí entity, se kterými systém komunikuje

- Procesy – transformují určité vstupy na výstupy
- Datové toky – znázorňují cestu, po které se pohybují datové shluky z jedné části systému do druhé
- Paměti – kolekce dat v klidu

Později přišli jiní pánové (Gane, Sarson) s vylepšením této metody s názvem ***Logické modelování***. Vývoj při něm probíhá takto:*(DFD, Paměti, ERD, Vazby, Překreslení, Jednotky, Popis)*

1. Vytvoří se systémový DFD, který zahrne procesy, datové toky, paměti a vnější entity. Vymezí hranice systému, vnější entity a datové toky na hranici systému. Ukáže uložená data a transformující procesy. Měl by být srozumitelný i zákazníkovi.
2. Načrtnutí datového modelu. Vytvoří se seznamy všech datových pamětí, které ponesou jména objektů, které uchovávají. Sledují se data, která do systému vstupují zvenčí až doputují do datových pamětí.
3. Analýza entit a vztahů – logický datový model. Vyhledáváme objekty – entity a hledáme mezi nimi vztahy. Vytváříme ERD.
4. Vytvoříme ERD v *normalizovaném tvaru*. Normalizovaných tvarů je větší množství. Neměly by tam být vazby M:N, dále by každá tabulka měla mít jeden klíč a všechny položky by měly záviset na klíči.
5. Překreslení DFD tak, aby reflektoval změny provedené v ERD.
6. Z procesů a dat vytvoř procedurální jednotky, které lze samostatně provozovat a naprogramovat – tj. rozdělit práci programátorům.
7. Specifikuj detaily každé jednotky např. DFD jednotky, popis ve strukturované angličtině, příklad výstupů atd.



Entitně relační diagram ERD: definuje neměnné atributy a strukturu dat. Datový model vyjadřuje vztahy, které nejsou zachyceny v procesních modelech. Komponentami datového modelu jsou:

- Entitní množiny
- Relace mezi entitami.

SW metriky a jejich využití.

SW metriky jsou měření některých z vlastností jaké má software. Bez měření nelze kvalitně řídit ani hodnotit kvalitu SW

Druhy metrik:

External – explicitní – zjistí se z artefaktů produktů systému (například délka programu, počet podprog., počet metod, rozsah a výskyt operandů, rozsah a výskyt operací, počet fcí, počet datových tabulek, ...)

Internal – implicitní – z procesů během vývoje (např doba řešení, velikost týmu v čase, spotřeba práce, produktivita, počet selhání systému v čase, defekty, spokojenost zákazníka, ...)

Potíže se metrikami:

- rozptyl hodnot
- produktivita práce programátorů, jejich kvalita
- druh SW, obtížně splnitelné termíny realizace
- moderní projekční techniky
- kvalita zúčastněných
- omezení SW a HW

Datové typy metrik:

- příslušnost ke třídě (číslo tramvaje), vztah: =
- fuzzy metrika – slabý, dobrý, velmi dobrý, vynikající, vztah: =, ≤
- fuzzy číselná – známky ve škole, vztah: =, ≤, (aritm. operace)
- intervalová (teplota)
- čas, vztah: =, ≤, interval, omezující operace
- plná data (délka)

Použití SW metrik

a) ***Výzkum:*** podklad pro **hledání metod realizace** softwarových produktů, které by přinesly podstatné zvýšení jeho kvality a snížení nákladů a doby vývoje a hlavně rozsahu prací při údržbě softwaru (výzkum metod a zákonitostí vývoje softwaru).

b) ***Normy:*** základ pro **stanovení technicko-ekonomických podkladů** pro řízení prací při tvorbě softwaru (normy pracnosti, odhady takových metrik, jako je pracnost či doba řešení) a uzavírání smluv (cena, termíny), předpoklad CMM

c) ***Kontrola kvality:*** prostředek **sledování spolehlivosti** softwaru při provozu a podklad pro řídící zásahy během údržby,procesy zajišťujíící kvalitu

d) **Operativa:**prostředek **sledování průběhu prací** při vývoji (dodržování termínů, procento testovaných komponent, trendy počtů chyb, počty nově zanesených chyb, komponenty s největším počtem chyb, atd.).

Výsledkem výzkumu SW metrik jsou metodiky vývoje (SOA, OO, strukturované programování, znalosti a vlivu kvality specifikací atd.) a metodiky odhadu pracnosti a doby řešení COCOMO a SOA ITIL ISO 9126, ISO 250xx, ISO 12207, ISO 20000

Funkční body

Normalizovaná metrika SW projektu. Je to metodika pro měření velikosti informačního systému. Počítají se funkce (činnosti) software a na základě jejich váženého součtu se provede odhad. Používa sa na odhad rozsahu softvérového systému, najmä pre interaktívne databázovo-orientované systémy (napr. obchodné aplikácie, angl. business applications). Nehodí sa pre systémy so zložitým vnútorným spracovaním. Počítá se například:*(Vstupy, výstupy, dotazy, soubory)*

- Počet vnějších vstupů a výstupů.
- Počet vnějších dotazů.
- Počet interních souborů
- Počet externích souborů.

Výsledek se zanese do formuláře FP, který se pak znásobí koeficientem složitosti. Výsledkem je odhad práce, času a velikosti.*(Práce, čas, velikost)*

Cocomo

(Typ projektu, počet řádků kódu)

Constructive cost model – založen na souboru významných veličin, které ovlivňují cenu a dobu řešení projektu. Výstupem je počet člověko-měsíců. Existuje základní ‚stredne a rozšířené COCOMO. Základní COCOMO používá rozdělení projektů na tři typy: organicky, prechodny a viazany. Pak používá metriku počet řádků kódu a na základě určení typu projektu z této metriky pomocí různých vzorečků odvozuje potřebný počet programátorů a dobu vývoje v měsících. Základním vstupem je odhad počtu řádků kódu v tisících. Z tohoto údaje se vypočítá EFFORT (člověko-měsíce), z něj potom DEVELOPMENT TIME (měsíce). Jejich podíl potom dává počet potřebných lidí. Při výpočtech se používají koeficienty z tabulky v závislosti třídě projektu.

Typy projektů:

1. Organický typ
   velikost projektu = málo set tisíc řádků, malé problémy pro malé týmy, typické jsou mírné normy a malá omezení na specifikaci rozhraní a možnost ovlivnit požadavky. Algoritmy a postupy jsou dobře známy, podmínky realizace relatvině stabilní, nejsou ostré podmínky ani termíny. (Příklad: zpracování dat v dávkovém provozu, úpravy známého operačního systému nebo kompilátoru)
2. Přechodný typ
   typ mezi organickým a vázaným, velikost projektu < 400 tisíc řádků, pokud není kód generován automaticky. V týmu jsou zkušení i méně zkušení pracovníci, tým má ne moc velké zkušenostiz předchozích obdobných realizací, úloha poměrně složitá (Příklad: menší dedikované oper. systémy, běžné transakční systémy, systém řízení výroby, jednoduchý systém řízení vojsk a zbraní)
3. Vázaný typ
   SW pracuje za velmi ostrych omezení na výkonnost a dobu odezvy, velikost jsou miliony řádků, vyžaduje práci s komplikovanými SW a HW systémy za ostrých předpokladů na předepsané fce, spolehlivost, termíny, přenositelnost, modifikovatelnost. Požadavky je obtížné měnit, řeší se nové problémy, se kterými se pracovníci dosud nesetkali. Obvykle: specifikaci požadavků + návrh dělá malá

skupina, vývoj částí – velký tým. Programování a testy – souběžně. (Příklad: nové rozsáhlé oper. Systémy, , kosmické lodě, letadla, atomové elektrárny, RT aplikace)

Toto by mali byt tie veliciny:

Faktory výrobku:
RELY - spolehlivost
DATA - rozsah dat (Jedná se o malou, střední, nebo rozsáhlou databázi ?)
CPLX - složitost výrobku, programu (Je to složitý systém ?)

Faktory počítače: TIME - časové omezení (Musí program běžet rychle ?)
STOR - paměťové omezení (Je dost paměti ?)
VIRT - stabilita počítače (Hotový nebo vyvíjený firmware ?)
TURN - rychlost odezvy počítače při vývoji

Faktory řešitelů:
ACAP - znalosti a zkušenosti analytika
AEXP - znalost aplikace
PCAP - zkušenosti a schopnosti programátorů (začátečník nebo zkušený)
VEXP - znalost virtuálního počítače (operační systém a program.prostředí)
LEXP - znalost programovacího jazyka

Faktory projektu:
MODP - moderní programovací metody (strukturované programování ?)
TOOL - použití program. nástrojů (nástroje CASE ?)
SCED - časový plán (volné nebo napjaté termíny ?)

Každý z těchto faktorů může příznivě nebo nepříznivě ovlivnit výsledný odhad. Hodnoty faktorů spadají do jedné z 5 kategorií, které charakterizují význam uvedeného aspektu v projektu:

velmi nízký, nízký, normální, vysoký, velmi vysoký

Kvalita SW

Kvalita je dodržení:

- explicitně stanovených funkčních a výkonových požadavků,
- standardů a charakteristik, které očekáváme od vyrobeného software.

Stupeň, do jaké míry SW splňuje specifikované požadavky nebo zákazníkovy potřeby a očekávání.

Kvalitu lze měřit:

- Přímo – např. počet chyb ve vztahu k počtu řádků kódu.
- Nepřímo – použitelnost, udržovatelnost.

Největším zdrojem defektů bývají samotné požadavky na systém.

Nejdráž se defekty odstraňují poté co je systém nasazen do provozu.

Techniky zajištění kvality

Jedná se o soubor technik patřící do společného oboru: SQA – software quality assurance.
Obecně lze mluvit o testování, validaci a verifikaci produktu.

- **Validací** se rozumí otázka, zda jsme vytvořili správný produkt tj. jestli produkt odpovídá potřebám uživatele (specifikaci), tj. „Dělat správné věci“.
- **Verifikací** se rozumí, zda jsme produkt vytvořili správně co se týče vnitřní činnosti, tj. „Dělat věci správně“.
- **Testováním** se pak pokoušíme zmíněné předchozí dokázat pro omezenou sadu příkladů. Obecně se pak za úspěšný považuje takový test, který odhalí nějakou chybu.

Testovat lze různými způsoby např.

- **Testování podle vstupních dat** – vstupní data se rozdělí do vhodných tříd a produkt se otestuje.
- **Testování podle struktury programu** – zde se snažíme o to, aby program prošel všemi možnými větvemi.

Jinou možností jsou **inspekce** (formální přezkoušení) produktu, která spočívá v dialogu mezi inspektory a autory, kde se kladou otázky nad produktem a specifikací s cílem nalézt případné chyby. Základem je nový pohled na projekt.Pro určité omezené oblasti lze použít i matematické důkazy správnosti programů.

ISO 9000

Obecně standardy rodiny ISO 900x se zaobírají systémem kvality při návrhu, vývoji, výrobě a servisu. Konkrétně ISO 9000 je doporučení jak aplikovat ISO 9001 při vývoji software. Samotný standard se postupně vyvíjel:

- V první verzi z roku 1987 se orientoval zejména na dodržování pracovních postupů.
- V druhé verzi z roku 1994 pak zejména na zajištění kvality a opět na držení se řádně dokumentovaných pracovních postupů.
- Ve třetí verzi z roku 2000 se standard zaměřil zejména na process management a jeho efektivnost.