Úvod

Software je tvořen celkovým souhrnem počítačových programů, procedur, pravidel a průvodní dokumentace a dat, která náleží k provozu počítačového systému.
Softwarový produkt je výrobek určen k předání uživateli.
Vyráběn **inženýrskými metodami**, **neopotřebuje** se fyzicky, obvykle **vyroben na míru** (málo produktů je sestaveno z předem existujících komponent).
Generické produkty - Samostatné systémy vytvářené výrobní organizací a prodávané na volném trhu libovolnému ze zákazníků, který si může výrobek koupit. Specifikace tvoří obchodní oddělení.
Smluvní, zakázkové, zákaznické produkty - Systémy objednané určitým zákazníkem. Software vyvíjí na zakázku smluvně vázaný dodavatel. Specifikace je součástí kontraktu mezi zákazníkem a dodavatelem.
Softwarové inženýrství je samostatný inženýrský obor, který řeší systematický přístup k vývoji, provozování, údržbě a nahrazování software.
Dobře řešený software = udržovatelnost + spolehlivost + efektivita + použitelnost
Udržovatelnost - Měl by být umožněn vývoj SW podle měnících se potřeb zákazníka.
Spolehlivost - Mezi atributy SW, na který je spolehnutí, patří spolehlivost, ochrana a bezpečnost. SW by neměl při výpadku systému způsobit fyzické, ani ekonomické škody.
Efektivita - SW by neměl plýtvat prostředky systému (paměť, čas procesoru a pod.).
Použitelnost - SW by měl mít přiměřené uživatelské rozhraní a odpovídající dokumentaci.
Kritické faktory SW produktivity = složitost (nepřímá charakterizace některými viditelnými atributy programu (počet proměnných) + velikost (programování v malém vs. velkém) + komunikace (jednotlivec, malý, velký tým) + čas, plán prací + neviditelnost SW
Programování v malém - ověřené techniky, shora-dolů, strukturované kódování, prostupné zjemňování, zdokonalování, inspekce logiky a kódu, nástroji jsou překladače a odvšivovače.
Programování ve velkém - plánovací mechanismy (dělení práce, harmonogram, zdroje), dokumentovaná specifikace, strukturovaný tým, formalizované soubory testů, testovací příklady, formalizované specifikace.
Proces vývoje SW - uživatelovy potřeby jsou transf. na požadavky na SW, ty jsou transf. na návrh, který je implementován jako kód a ten je testován, dokumentován a certifikován pro operační použití.

Základní aktivity při vývoji SW

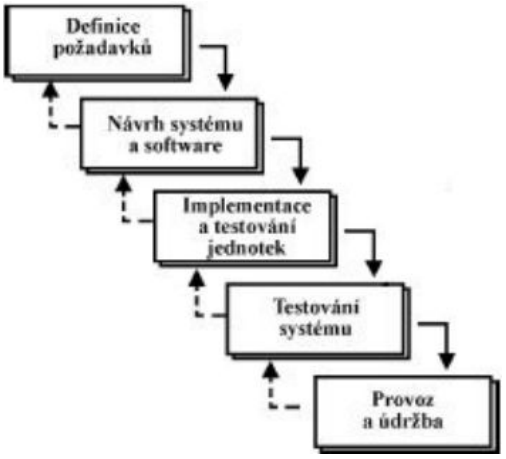
Specifikace - je třeba definovat funkcionalitu SW a operační omezení.
Vývoj - je třeba vytvořit SW, který splňuje požadavky kladené ve specifikaci.
Validace - SW musí být validován („kolaudován“), aby bylo potvrzeno, že řeší právě to, co požaduje uživatel.
Evoluce - SW musí být dále rozvíjen, aby vyhověl měnícím se požadavkům zákazníka.

Viditelné znaky výroby SW jsou **artefakty** (výpisy programů, dokumentace, data, zdrojové soubory) a **procesy** (pracovní postupy, uznávaná pravidla, interakce mezi členy týmu).

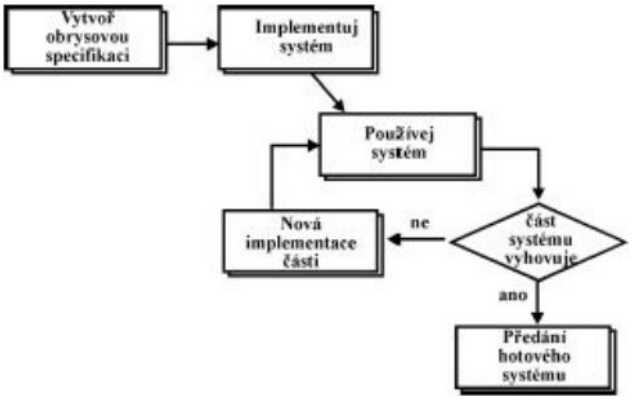
Charakteristiky výrobního procesu SW

Srozumitelnost - je proces explicitně definován a je snadné porozumět definici procesu?

Viditelnost - vyúsťují procesní aktivity ve zřetelné výsledky tak, že postup procesu je viditelný zvenčí?
Spolehlivost - je proces navržen tak, že se lze chybám procesu vyhnout, nebo je zachytit dříve, než zaviní chyby ve výrobku?
Přijatelnost - je definovaný proces přijatelný a použitelný inženýry zodpovídajícími za produkci?
Robustnost - může proces pokračovat i v po výskytu neočekávaných problémů?
Udržovatelnost - může se proces vyvíjet tak, aby odrážel měnící se organizační požadavky nebo identifikovaná zlepšení procesu?
Rychlost - jak rychle lze realizovat výrobní proces, který z dané specifikace vytvoří hotový systém předaný zákazníkovi?
Podporovatelnost - do jaké míry lze aktivity procesu podpořit nástroji CASE?

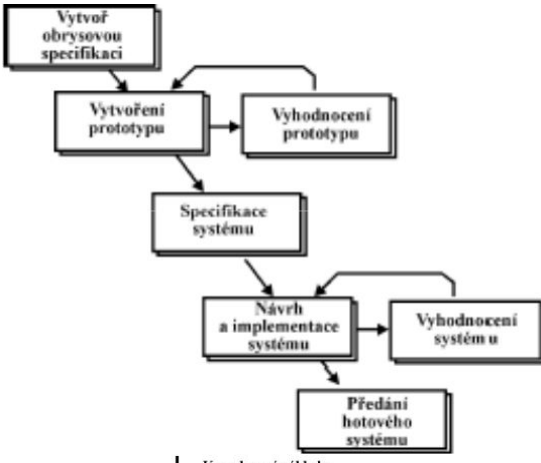


Model ž. c. vodopád - Integrace systému zároveň s jeho testy, problémy s cenou v případě nezdaru některé z pozdějších etap.
Problémy vodopádu
- Reálné projekty nedodržují jednotlivé kroky v předepsaném pořadí.
- Uživatel nedokáže v počátečních etapách formulovat požadavky na systém přesně.
- Zákazník musí být trpělivý. Pozdní odhalení nedostatků může vážně ohrozit celý projekt.
(Manažeři tento model preferují.)
Výhody vodopádu
- Klade stejný důraz na dokumentaci jako na zdrojový kód.
- Vždy je jasné, jaký další krok následuje - usnadnění pro management.
- Integrace systému zároveň s jeho testy.
Vývoj podle obrysové specifikace (vodopádu) - vyžadování vysoce talentovaní pracovníci (průměrný tým ne), malý tým, systémy jsou obvykle špatně strukturované (neustálé změny poškozují strukturu systému, evoluce obtížní a nákladná), proces není viditelný.

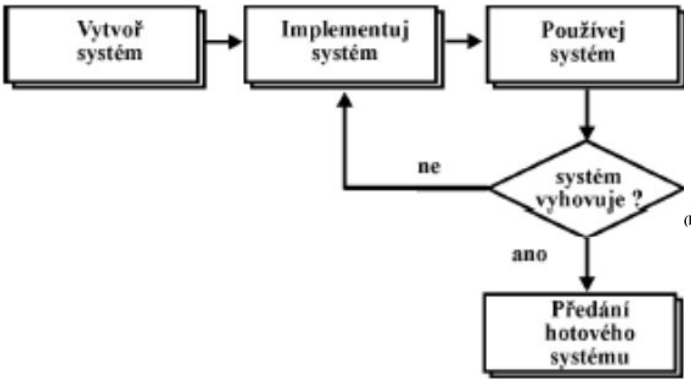


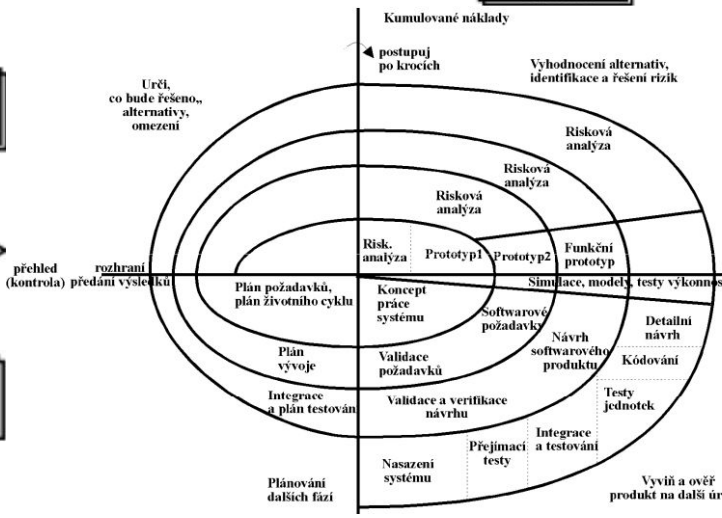
Model inkrementálního ž. c.
Problémy
- obrysová specifikace vs. realita
- dokumentace programu vs. specifikace
- údržba zvyšuje entropii.
Výhody
- Zákazník může průběžně dávat feedback ke stavu softwaru - vývojáři tak získají cenné informace o používání či chybách.
- Vzhledem k tomu že inkrementální model implementuje software po částech, jsou pozdější úpravy modulů poměrně jednoduché.
- Software je použitelný už ve své rannější fázi, ne až po dokončení všech vývojových fází jako je tomu např. u vodopádu.

Rozdíl mezi iterativním a inkrementalním vývojem
- **Inkrementální vývoj** využívá daný počet kroků a vývoj tak jde od začátku do konce lineární cestou. Kroky inkrementálního vývoje mohou být: návrh, implementace, testování, údržba. Ty se dají dále rozdělit. Klasickým příkladem je model vodopád.
- **Iterativní přístup** nemá pevně daný sled kroků, vývoj probíhá v cyklech. Méně se zabývá sledováním postupu u jednotlivých funkcionalit. Je zaměřen na vytvoření funkčního prototypu a nabalování další funkcionality v jednotlivých cyklech. Příkladem je agilní vývoj.



Model ž. c. prototypování - tvorba prototypů probíhá za účelem získávání poznatků, po specifikaci je prototyp zapomenut.
Model ž. c. výzkumník - problémy - obtížné manažerské řízení, neexistující a neplatná dokumentace, nenahraditelnost řešitelů. Je to experimentování, u kterého často netušíte, jak to dopadne.





Brooksův zákon - přidání řešitelské kapacity u opožděného projektu může zvětšit jeho zpoždění. Náklady na začlenění nového člena do týmu jsou větší než jeho přínos. Stávající programátor se musí věnovat novému členovi místo své produktivitě na projektu.

Lehmanovy zákony

Z. trvalé proměny: Systém používaný v reálném prostředí se neustále mění, dokud není levnější systém restrukturalizovat, nebo nahradit zcela novou verzí.
Z. rostoucí složitosti: Při evolučních změnách je program stále méně strukturovaný a vzrůstá jeho vnitřní složitost. Odstranění narůstající složitosti vyžaduje dodatečné úsilí.
Z. vývoje programu: Rychlost změn globálních atributů systému se může jevit v omezeném časovém intervalu jako náhodná. V dlouhodobém pohledu se však jedná o seberegulující proces, který lze statisticky sledovat a předvídat.
Z. invariantní spotřeby práce: Celkový pokrok při vývoji projektů je statisticky invariantní. Jinak řečeno, rychlost vývoje programu je přibližně konstantní a nekoreluje s vynaloženými prostředky.

Z. omezené velikosti přírůstku: Systém určuje přípustnou velikost přírůstku v nových verzích. Pokud je limita překročena, objeví se závažné problémy týkající se kvality a použitelnosti systému.

Agilní metodiky

Manifest a. programování - umožnit změnu je mnohem efektivnější, než se jí snažit zabránit. Je třeba být připraven reagovat na nepředvídatelné události, protože ty nastanou.
- **Individuality a interakce** mají přednost před procesy a nástroji.
- **Fungující software** má přednost před obsáhlou dokumentací.
- **Spolupráce se zákazníkem** má přednost před sjednáváním smluv.
- **Reakce na změnu** má přednost před plněním plánu.

V **tradičním přístupu** je funkcionalita a flexibilní je čas a cena, v **agilním přístupu** je to naopak.

Společné rysy agilních metodik

Iterativní a inkrementální vývoj s krátkými iteracemi - vývoj probíhá po krátkých fázích, takže celková funkcionalita je dodávána po částech, zákazník tak má možnost průběžně sledovat vývoj a může se k němu vyjádřit a oponovat změny, zákazník má na konci jistotu, že nedostane, co neočekával.
Komunikace mezi zákazníkem a vývojovým týmem - ideálně je členem vývojového týmu, má možnost okamžitě vidět průběžné výsledky a reagovat na ně, zákazník se účastní sestavování návrhu, spolurozhoduje o testech a poskytuje zpětnou vazbu vývojářům.
Průběžné automatizované testování - díky rychlým iteracím se aplikace mění rychle, je proto nutné pro zajištění kvality ověřovat její funkčnost průběžně, testy by měly být automatizované, předem sestavené a napsány před implementací testované části, při každé změně musí být aplikována kompletní sada testů, aby nebyla porušena integrace jedn. částí.

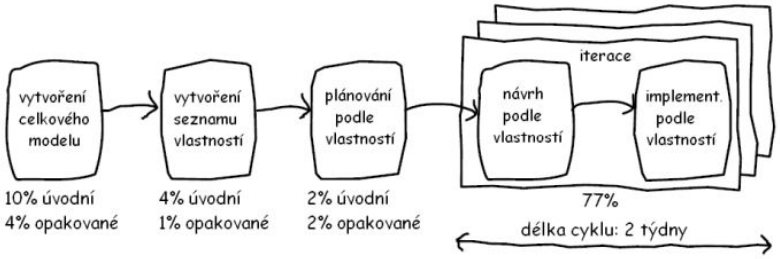
Společné rysy generických a agilních vývojových metodik

- Iterativní a inkrementální vývoj s krátkými iteracemi
- Komunikace mezi zákazníkem a vývojovým týmem
- Průběžné automatizované testování

Extreme programming - pro menší projekty a malé týmy, vyvíjející SW podle nejasného nebo rychle se měnícího zadání. Jediným exaktním zdrojem informací je zdrojový kód. Používá principy a postupy, které dotahuje do extrému (např. neustálé testování).
Principy XP
- **Rychlá zpětná vazba**, která spočívá v rychlém zjištění stavu systému po provedené akci, vyhodnocení akce a uložení výsledku vyhodnocení co nejdříve zpět do systému.
- **Předpoklad jednoduchosti**, představuje v mnoha ohledech nejobtížnější princip,protože je v protikladu s tradičním pojetím programování, kdy se vše plánuje a navrhuje do budoucna tak, aby to bylo znovu použito. XP naopak předpokládá u řešitelského týmu schopnost přidat složitost tam, kde to bude v budoucnu účelné.
- **Přírůstková změna**, vychází z předpokladu, že velké změny provedené najednou nefungují. Veškeré změny v projektu se proto provádějí pomocí malých přírůstků.

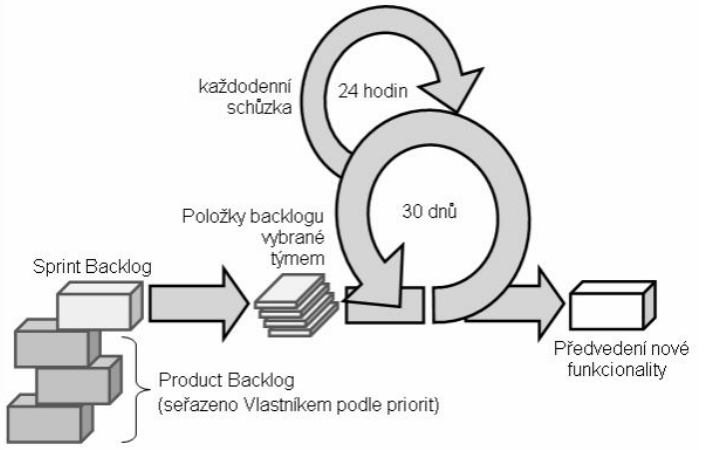
- **Využití změny**, vychází z předpokladu, že nejlepší strategie je ta, která si zachová co nejvíce možností řešení nejnaléhavějších problémů projektu.
- **Kvalitní práce**, která představuje fixní proměnnou ze čtyř proměnných pro posouzení projektu (šíře zadání, náklady, čas a kvalita) s hodnotou vynikající, při horší hodnotě členy týmu práce nebude bavit a projekt může skončit neúspěchem.

Feature-Driven Development



Vývoj po malých kouscích (rysech), což jsou elementární části funkcionality přinášející nějakou hodnotu uživateli. Vývoj probíhá v 5 fázích, první 3 jsou sekvenční, poslední 2 iterativní. Iterace = 2 týdny. Měří pokrok ve vývoji projektu, FDD detailně plánuje a kontroluje vývojový proces, zaměření na dodávání fungujících přírůstků každé 2 týdny.

SCRUM Development Process



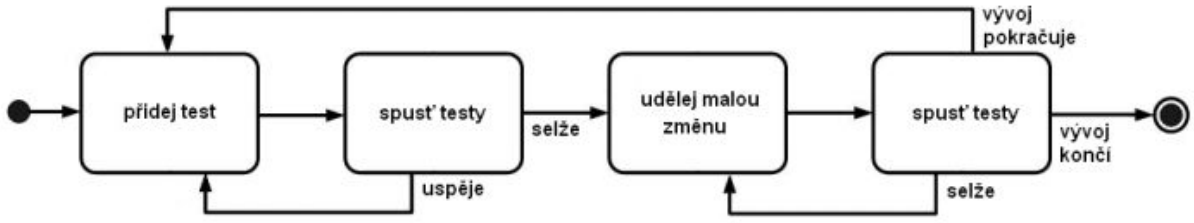
Principem je iterativní vývoj o 3-8 fázích (sprinty), každý trvá měsíc. Nedefinuje žádné konkrétní procesy, pouze zavádí pravidelné každodenní schůzky vývojového týmu, kde si sdělují, co bylo od minula dokončeno, co se bude dělat teď a jaké jsou překážky vývoji v cestě. Každý sprint zakončen předvedením funkční demo-aplikace zákazníkovi, který poskytne zpětnou vazbu.

Co je to product backlog?

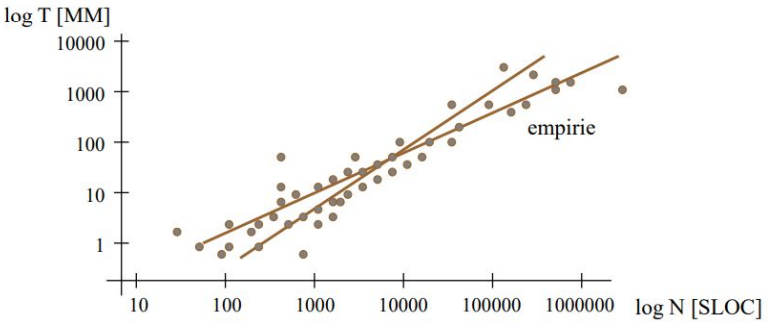
Používá se u Scrumu. Je to jednoduše seznam věcí které se na projektu musí udělat. Každá položka obsahuje odhad potřebného času na dokončení a prioritu. Nahrazuje klasickou specifikaci požadavků. Položky backlogu mohout mít jak technickou (přidat tuto funkcionalitu, fixnout tento bug) tak uživatelskou povahu (zjednodušit uživatelské rozhraní). Product owner ho používá při plánování sprintu k zadávání úkolů.

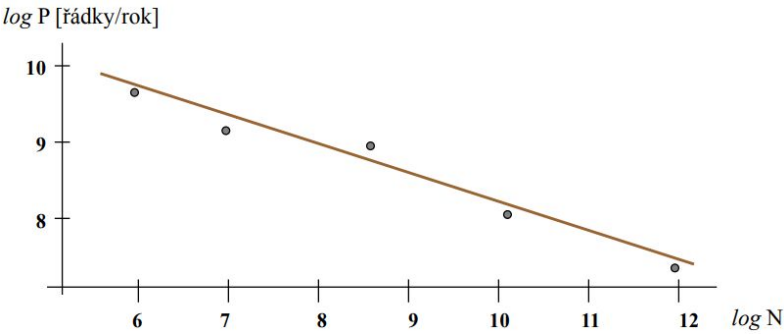
Test-Driven Development

Nezabývá se tvorbou specifikací, plánu a dokumentace, to si každý tým musí zvolit sám. Doporučuje přistoupit k testům jako k hlavní fázi celého vývojového procesu. Základním pravidlem je psát testy dříve než samotný kód a implementovat jen přesně ty části kódu, které projdou testem.



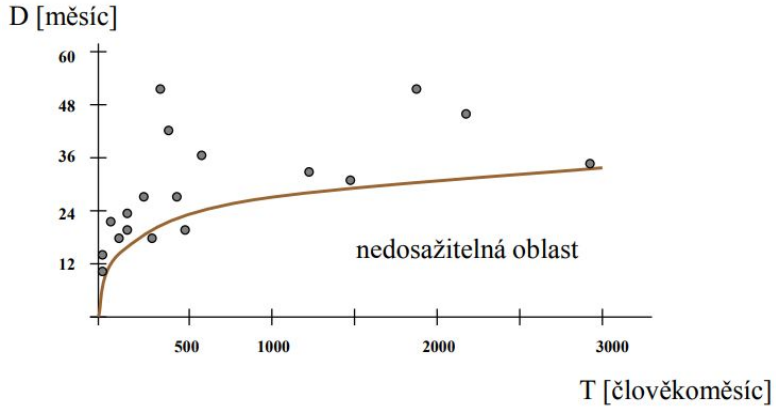
SW fyzika

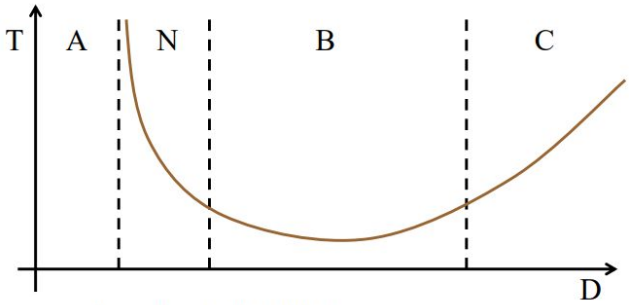




S rostoucí délkou programu roste spotřeba práce. S roustocí dělkou programu klesá produktivita programátorů.

Putnamova rovnice - $N = c.T^{1/3} * D^{4/3}$
N - délka programu (počet řádků, SLOC), c - škálovací faktor velikosti projektu, T - spotřeba práce (člověkoměsíc), D - doba realizace projektu.
Používá se k empirickému odhadu úsilí vývoje softwaru.
Důsledky - Programy psané ve spěchu jsou delší. Při zkrácení termínu na 83 % je pracnost dvojnásobná.





A - nedosažitelná oblast
N - napjaté termíny
B - oblast stability
C - doba řešení je zbytečně dlouhá

Cocomo - odhadování ceny SW

Je to model používaný k odhadování ceny SW.
Idea - Cena vývoje aplikace přímo závisí na velikosti SW. Přednost odhadu velikosti SW závisí na etapě vývoje. V pozdějších je přednější. Přesnost odhadu se může lišit až 4x oběma směry.
Zdroje empirických dat - větší počet předchozích komplexních projektů (aplikace odlišného druhu s odlišnými cíly, odlišná vývojová prostředí), rozhovory s více manažery. Parametry modelu byly nastaveny podle získaných empirických dat.
3 úrovně detailu
Základní - hrubý odhad E a T založen na odhadu KSLOC
Střední - vliv jiných faktorů na E a T
Pokročilý - bere v úvahu vlivy vývojové etapy, ve které se projekt nachází

3 vývojové módy
Organický - jednodušší, dobře řešitelné projekty, menšího rozsahu
Bezprostřední - středně obtížné projekty
Vázaný - rozsáhlé projekty s vysokými nároky na řízení
Úsilý E - $E = a * (KSLOC)^b$
Doba vývoje T - $T = c * E^d$
Parametry a, b, c, d jsou voleny dle modelu a módu. **a** je ovlivněn Korekčním faktorem $\mathbf{F_c}$ pro střední a pokročilý model. Ostatní jsou konstantní.
Korekční faktor F_c je součinem hodnot 15 atributů specifických pro vývojový proces. Každý nabývá 6 možných hodnot (velmi nízký, nízký, normální, velký, velmi v., extrémně v.). Upravuje COCOMO na základě množiny subjektivních faktorů.
Atributy mající vliv na F_c
- **atributy SW produktu** - požadovaná spolehlivost, velikost databáze, složitost produktu
- **HW atributy** - omezení času výpočtu, využití paměti/disku, spolehlivost
- **atributy vývojového týmu** - schopnost analytická, programátorská, zkušenost s pod. apli.
- **atributy projektu** - použití moderních programovacích technik, použití SW nástrojů, přesné plánování.

Kroky COCOMO - Určení nominálního úsilí E_n, Určení korelačního faktoru F_c, Určení aktuálního (zpřesnělého) úsilí E, Určení doby vývoje T a dalších faktorů relevantních pro projekt.
COCOMO pro odhad nákladů při modifikaci existujících aplikací
ESLOC = ASLOC . (0,4 DM + 0,3 CM + 0,3 IM) / 100

ESLOC - ekvivalentní počet SLOC
ASLOC - odhadnutý počet modifikovaných SLOC
DM - procento modifikace v návrhu
CM - procento modifikace v kódu
IM - integrační úsilí (procento původní práce)

Cocomo 2

Potřeba změnit COCOMO - nové SW procesy, nové jevy měření velikostí, nové jevy znovupoužití SW, potřeba rozhodování na základě neúplné informace

3 modely

ACM (Application Composition Model) - pro projekty s použitím moderních nastrojů a GUI
EDM (Early Design Model) - pro hrubé odhady v úvodních etapách, kdy se architektura vyvíjí
PAM (Post Architecture Model) - pro odhady poté, co byla specifikována architektura

Early Design a Post-Architecture
$PM = A * (Size)^{(SF)} * (EM)$
Velikost určena několika přístupy - KSLOC, UFP (neupravené funkční body), EKSLOC (ekvivalentní velikost zdrojového kódu)
SF - měřítkové faktory určené pomocí driverů exponentu
EM - multiplikátory úsilí (7 pro ED, 17 pro PA)

Nové atributy ovlivňující EM - např: složitost HW platformy, proměnlivost HW platformy, personální schopnosti a zkušenosti, bezpečnost. Většina vychází z kombinace dříve používaných atributů.
COCOMO pro odhad nákladů při modifikaci existujících aplikací
ESLOC = ASLOC . (AA+SU+0,4 DM+0,3 CM+0,3 IM)/100

AA - práce potřebná pro určení, zda a v jakém rozsahu může být existující modul použit beze změn, SU = čitelnost a uchopení

2 společné vlastnosti C a C2
- Oba způsoby při odhadu ceny zahrnují jistou množinu faktorů, která ji ovlivňuje
- Oba využívají stejný druh modelů na rozlišení výpočtu

3 rozdíly C a C2
- COCOMO 2 zahrnuje některé nové atributy na měření odhadu ceny, které vznikly kombinací předchozích z COCOMO.
- Modely v COCOMO 2 jsou na rozdíl od COCOMO zaměřené spíše na vývojovou etapu projektu
- Při odhadu nákladů na úpravu aplikace využívá COCOMO 2 i tzv. AA a SU koeficienty.

Funkční body

(Pro výrobu SW je třeba určit jednotku výroby a cenu práce za její výrobu.)
Funkční body = normalizovaná metrika SW projektu
Měří aplikační oblast, nezkoumá technickou oblast. Měří aplikační funkce a data, neměří kód
Je to předběžný odhad s použitím omezené informace. měří vstupy, výstupy, dotazy, vnitřní a vnější paměti. Princip odhadu = velikost projektu * složitost * rizikové faktory

Typy funkčních bodů

FB vztažené k transakčním funkcím: Externí vstupy, externí výstupy, externí dotazy.
FB vztažené k datovým funkcím: Interní logické soubory, soubory vnějšího rozhraní.

Externí vstupy
Započteme každá unikátní uživatelská data nebo zadání uživatelských povelů, která vstoupí přes externí rozhraní do aplikace a přidá, mění, ruší nebo jinak pozmění data (např. přiřazení, přemístění, ...) v interním logickém souboru.

Externí výstupy
Započteme každá unikátní uživatelská data nebo řídící data, která opouští externí hranici měřeného systému. Externí výstup je považován za unikátní, pokud má odlišná data, nebo pokud vnější návrh (jiná aplikace) vyžaduje odlišnou logiku zpracování oproti jiným externím výstupům.

Externí dotazy
Jako vnější dotaz započti každou unikátní vstupně/výstupní kombinaci, kde vstup je příčinou a generuje výstup. Vnější dotaz je považován za unikátní, pokud se od ostatních dotazů

odlišuje typem výstupních datových elementů, nebo pokud vyžaduje odlišnou logiku zpracování v porovnání s ostatními externími dotazy.

Interní logické soubory

Každá velká logická skupina uživatelských dat nebo informací použitých pro řízení aplikace představuje jeden ILF. Zahrneme každý logický soubor, nebo v případě DB, každé logické seskupení dat z pohledu uživatele, které je vytvořeno, používáno, nebo udržováno aplikací.

Spíše než fyzické soubory započteme každé logické seskupení dat tak, jak je viděno z pohledu uživatele a jak je definováno při analýze požadavků nebo návrhu dat.Nezapočteme soubory, které nejsou přístupné uživateli prostřednictvím vnějšího výstupu nebo dotazu a které nejsou nezávisle udržovány.

Soubory vnějšího rozhraní

Započteme každou velkou logickou skupinu uživatelských dat nebo řídící informace používané aplikací. Tato informace musí být udržována jinou aplikací. Zahrňte každý logický soubor nebo logickou skupinu dat z pohledu uživatele.

Započteme každou velkou logickou skupinu uživatelských dat nebo řídící informace, která je extrahována aplikací z jiné aplikace ve formě souboru externího rozhraní. Extrakce nemá mít za následek změnu v některém z interních logických souborů. Pokud ano, pak započteme do EI místo do EIF.

Jak dostaneme z neupravených funkčních bodů počet upravených

- Před výpočtem musíme EI, EO, EQ, ILF, EIF roztřídit do skupin podle vah (nízká, průměrná vysoká). Roztřiďuje se dle matic.
- Dále existuje 14 charakteristik hodnocených podle stupně vlivu na aplikaci (hodnoceny 0 bez vlivu - 5 podstatný)
například: Je vnitřní zpracování složité? Jsou vstupy, výstupy, soubory a dotazy složité? Jsou hlavní soubory opravovány on-line?
- počet funkčních bodů = (0.65 + (0.01 * součet hodnocení charakteristik systému)) * počet neupravených FB.

Rozdíl mezi FB projektu a produktu

- **Funkční body produktu** - při vývoji aplikace na zelené louce jsou tyto body takové body, které zůstanou v aplikaci na konci vývoje
- **Funkční body projektu** - prošly týmu rukami, zaplatili jsme za ně, nemusí na konci vývoje zůstat (např. body vynaložené na vyzkoušení nějakého postupu a následné předělání zpět), musíme s nimi však nutně počítat.

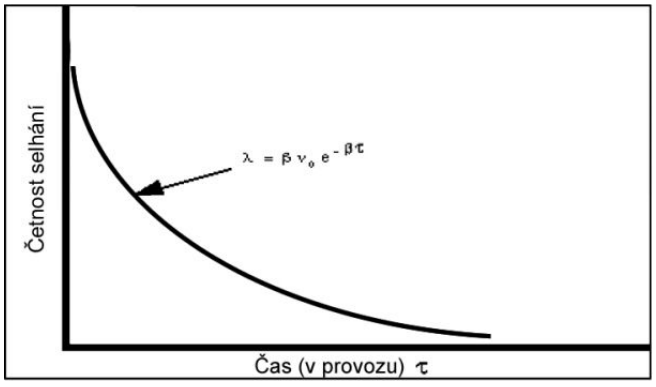
Rozdíl při počítání funkčních bodů u finálního SW produktu a při kalkulaci cen za SW projekt:

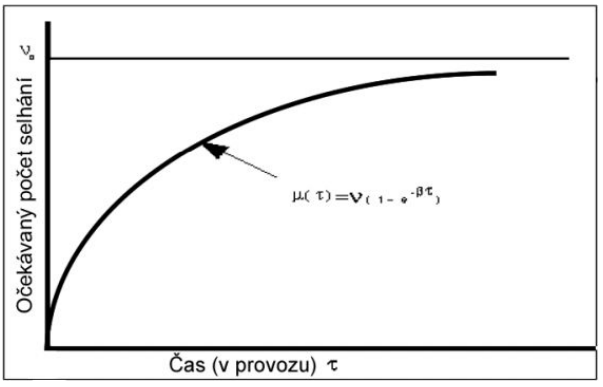
Type of Project	Project Function Points	Application Function Points Installed Function Pts. (IFP)
Development Project	Project FP = New (Added) FP + Conversion FP	Application FP = New (Added) FP
Enhancement Project	Project FP = Added FP + Changed FP + Deleted FP + Conversion FP	Application FP = Original FP - Deleted FP + Added FP + Δ Changed FP

Chyby SW

Nejvíc chyb na začátku (požadavky), nejméně později (kód), nejlevnější testování na začátku (požadavky), nejdražší na konci (integrační a systémové testy), cena odstranění je také nejlevnější na začátku a nejdražší v provozu.
Úspěšný projekt - dokončen včas, bez překročení rozpočtu, se všemi specifikovanými rysy a funkcemi
S výhradami - dokončen a funkční, překročil rozpočet, čas méně rysů a funkcí než specifikováno
Neúspěšný - zastaven před dokončením, neimplementován, vyřazen po instalaci

Prevence proti neúspěchu projektu
ZAPOJENÍ vrcholového řízení a koncových uživatelů
POUŽITÍ efektivního řízení projektu se spoluúčastí a zapojením vrcholového řízení na přezkoumáních
POUŽITÍ efektivního řízení požadavků
POUŽITÍ inkrementálního vývoje
UČINĚNÍ “všech smysluplných kroků” při inženýrských aktivitách, t.j. dokumentace, měření, plánování, sledování, řízení kvality…





Porucha - neschopnost systému nebo systémové komponenty provádět požadovanou funkci ve specifikovaných hranicích. Porucha může nastat, když se narazí na chybu, jejímž výsledkem je ztráta očekávané uživatelské služby.

Četnost chyb - dané kategorie nebo k dané jednotce měření
Fault (chyba) - … v kódu může být příčinou jednoho nebo více selhání, náhodná podmínka
Error - chyba (omyl), nesprávná nebo chybějící akce uživatele zapříčiní chybu v programu

IBM ortogonální klasifikace defektů (ODC)

- **Funkce** - chyba ovlivňující schopnosti, rozhraní uživatelů, rozhraní výrobku, rozhraní s HW architekturou nebo globální datovou strukturou.
- **Rozhraní** - chyba při interakci s ostatními komponentami nebo ovladači přes volání, makra, řídící bloky nebo seznamy parametrů.
- **Ověřování** - chyba v logice programu, která selže při validaci dat a hodnot před tím, než jsou použity.
- **Přiřazení** - chyba při inicializaci datové struktury nebo bloku kódu.
- **Časování/serializace** - chyba, která zahrnuje časování sdílených a RT prostředků.
- **Sestavení/balení/spojování** - chyba související s problémy s repozitory projektu, změnami vedení, nebo správou verzí.
- **Dokumentace** - chyba, která ovlivňuje publikace a návody pro údržbu.
- **Algoritmus** - chyba, která se týká efektivity nebo správnosti algoritmu nebo datové struktury, ne však jejich návrhu.

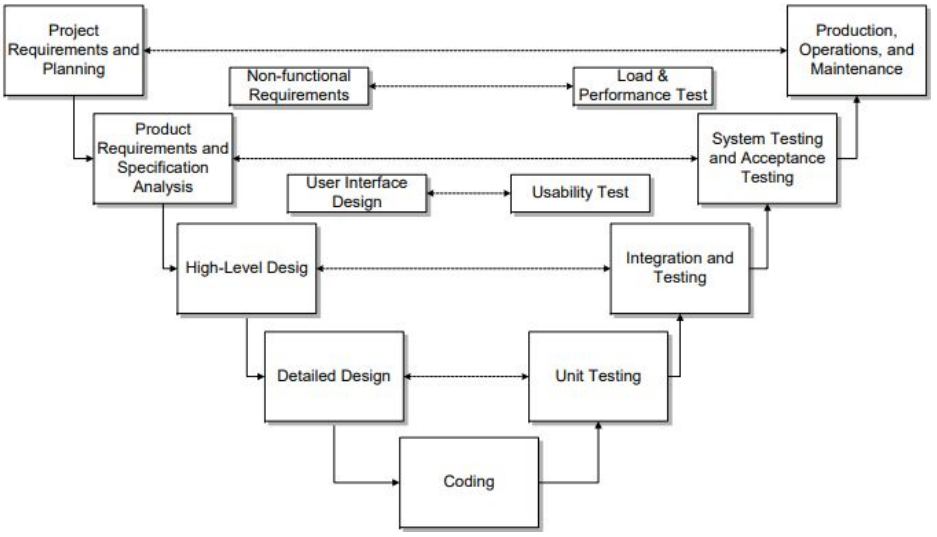
Typ defektu a asociace s etapou
Funkce (návrh), rozhraní (návrh na nízké úrovni), ověřování (návrh na nízké úrovní nebo kód), přiřazení (kód), časování/serializace (návrh na nízké úrovni), sestaveni/b/s (publikace), algoritmus (návrh na nízké úrovni)

Testování SW produktů

Testování je proces spouštění programu s cílem nalézt chyby. Dobrý testovací případ má vysokou pravděpodobnost nalezení dosud nenalezené chyby. Úspěšný test je takový, který odhalí neodhalenou chybu.
Testování nemůže ukázat nepřítomnost chyb, může jen ukázat, že v SW jsou chyby.
Ukazuje také funkce a výkon, je ukazatelem kvality SW.

V-model

- Rozšíření modelu vodopád. Vývoj neprobíhá v lineárním směru - křivka je ohnuta do tvaru V, po fázi implementace nastávají fáze testování, verifikace, validace a údržby.
- Model demonstruje vztah mezi každou fází návrhu a s ní asociovanou fází testování.
- Horizontální křivka reprezentuje dokončenost projektu, vertikální zase úroveň abstrakce.



Validace = test proti specifikovaným funkcích (dělat správné věci)
Verifikace = test proti vnitřní činnosti (dělat věci správně)
Úplné testy - průchod všech možných cest programu
Selektivní testy - testování důležitých cest, validuje rozhraní a vytváří důvěru ve vnitřní činnost SW, bílá skříňka
Dynamické testy - provedení programu s předem určenými vstupy, porovnání výstupů s očekávanými výsledky, nemůže prokázat absenci defektů, je to vzorkování, blackbox
Testovací případy - klíčové položky plánu testování, mohou obsahovat skripty, data, kontrolní seznamy.
Black box - test, zda všechny motory pracují. Nezajímá nás, jak to pracuje, ale co to dělá, zaměření na vstupy a výstupy.
White box - test činnosti každé funkce, zohledňuje strukturu programu, pokrytí průchodů kódem. Vypočítává se cyklomatická složitost (počet rozhodnutí + 1)
Vývoj/integrace/testování je místo, kde dochází nejčastěji k překrývání aktivit.

Integrační postupy
- **Shora dolů (TDT)** - nejprve je implementována kostra systému. Zkombinováno do minimální skořápky systému. Pro doplnění neúplných částí se použijí protézy nahrazované postupně aktuálními moduly.
    - **nevýhody** - Složité objekty, moduly nelze jednoduše zaměnit za protézu. Výsledky testů na vyšších úrovních nemusí být přímo viditelné.
- **Zdola nahoru (BUT)** - začne s individuálními moduly a sestavuje zdola. Individuální jednotky (po testování jednotek) jsou kombinovány do subsystémů. Subsystémy jsou kombinovány do celku.
    - **nevýhody** - Čas a náklady na konstrukci drivers pro testování jsou obykle vyšší než u protéz. Až v závěru vznikne program použitelný pro předvedení, ve formě prototypu.

Kalkulace projektu

Položky
- Stanovení popisu projektu, požadavků, doby trvání
- Stanovení činností a pracnosti (programování, testy, tvorba webu, zajištění bezpečnosti, školení, dokumentace, technická podpora)
- Stanovení rolí v projektu (PM, analytik, developer, tester,...), pro každou roli popis, náklady/MD a výnosy/MD (hodnota zaměstnance), finanční kategorie v rámci firmy.
- Tabulka činnost × role = pracnost (v MD)
- Další náklady: licence na SW, nákup HW, subdodávky, podpora, rizika
- Stanovení rizik (ID, popis, pravděpodobnost, dopady, náklady)
- Sumarizace: náklady, marže, výnosy

Základní body SLA
Účel smlouvy, doba trvání smlouvy, specifikace servisní činnosti, vymezení pojmů (např. školení uživatelů, aktualizace dat, provoz podpory), forma servisní činnosti, způsob oznamování závad, reakční doby a způsob odstranění závady, součinnost objednatele.

Definice SLA

SLA označuje smlouvu sjednanou mezi poskytovatelem služby a jejím uživatelem. Poměrně obšírně se tématem poskytování IT služeb zabývá metodika ITIL. Každé prodání (poskytnutí) produktu (software, služby, výrobku,...) je provedeno zároveň s definicí toho produktu. Tato definice by měla uživateli i výrobci vymezit jasná pravidla, jak se o produkt dále starat, jak jej používat a jeho cenu. Například se jedná o záruku na produkt, běžné užití produktu, k čemu je produkt určen, kolik produkt stojí,...

Inspekce

Efektivita roste se zvyšující se formálností. Inspekce může být **formální** (recenze, inspekce) a **neformální** (konverzace, neformální prezentace).
Zkušební, inspekční a recenzní tým - vedoucí týmu (moderátor), zapisovatel, autor, recenzenti a inspektoři.
Příprava inspektora - musí tomu rozumět, projít všechny materiály výrobku pro pochopení místa a formátu informace, opatřete poznámky, připomínky formulovat jako otázky, nehodnotit styl, informovat vedoucího recenze, pokud se nemůžete připravit.
Cíle formálního přezkoušení - odhalit chyby ve funkci, logice, implementaci SW. Ověřit, že zkoumaná položka splňuje požadavky. Zajistit, že položka byla prezentována s použitím předdefinovaných standardů. Zajistit jednotný vývoj. Zvýšit řiditelnost projektů.
Závažnost chyb - **kritické** (způsobí pád systému, chybné výstupy, není známa cesta vyhnutí se tomuto problému), **vážné** (chybné výstupy, je známa cesta, zasažena významná část systému), **středně závažné** (ovlivňuje omezenou část funkcionality, dá se vyhnout), **málo závažné** (mohou být opomenuty bez narušení funkčnosti).
Výhody formuláře pro recenzi - filtrování nepodstatných problémů před recenzí, problémy mohou být diskutovány v pořadí důležitosti, recenzenti specifikují problémy zřetelně během přípravy (úspora času)
Nevýhody formuláře pro recenzi - čas na přípravu, odrazuje od přípravy na poslední chvíli, zviditelňuje kvalitu přípravy

Kvalita SW produktů

Dodržení explicitně stanovených funkčních a výkonových požadavků, dodržení explicitně dokumentovaných vývojových standardů a implicitních charakteristik, které jsou očekávány u profesionálně vyrobeného software.
Aspekty kvality - odchylky od požadavků na SW, nedodržení standardů, odchylky od běžných zvyklostí
Přímo (počet chyb, KLOC, čas) a **nepřímo** (použitelnost, udržovatelnost) **měřitelné faktory**
Kategorie faktorů kvality - operační charakteristiky, schopnost akceptovat změny, adaptibilita na nové prostředí
Faktory kvality McCall

Korektnost - Rozsah toho, jak program splňuje specifikaci splňuje uživatelovy záměry.
Spolehlivost - V jakém rozsahu lze očekávat, že program bude plnit zamýšlené funkce s požadovanou přesností.
Efektivita - Množství výpočetních prostředků a kódu, které program potřebuje na splnění svých funkcí.
Udržovatelnost - Úsilí vyžadované na vyhledání a opravu chyby v programu.
Flexibilita - Úsilí vyžadované na modifikaci provozovaného programu.
Testovatelnost - Úsilí potřebné na testování programu tak, abychom se ujistili, že plní zamýšlené funkce.
Přenositelnost - Úsilí potřebné na přemístění programu na jiný HW/SW.
Znovupoužitelnost - Rozsah, v jakém lze program nebo jeho části znovu použít v jiné aplikaci (funkce a balení produktu).
Schopnost spolupráce - Úsilí, které je nutné vynaložit pro připojení daného systému k jinému.

CMM - Capability Maturity Model
Úroveň 1: Výchozí - Chaotický proces, nepředvídatelná cena, plán a kvalita.
Úroveň 2: Opakovatelný - Intuitivní; cena a kvalita jsou vysoce proměnlivé, plán je pod vědomou kontrolou, neformální metody a procedury. Klíčové prvky: řízené požadavky, plánování softwarového projektu, řízené subkontrakty na software
Úroveň 3: Definovaný - Orientován na kvalitu; spolehlivé ceny a plány, zlepšující se, ale dosud nepředvídatelný přínos (výkon) systému kvality. Klíčové prvky: zlepšování organizačního procesu, definice organizačního procesu, školící program
Úroveň 4: Řízený - Kvantitativní; promyšlená statisticky řízená kvalita produktu. Klíčové prvky: měření a kvantitativní řízení procesu výroby, řízení kvality
Úroveň 5: Optimalizující - Kvantitativní základ pro kontinuální investice směřující k automatizaci a zlepšení výrobního procesu. Klíčové prvky: prevence chyb, inovace technologie, řízené změny výrobních procesů

Metriky a měření

Míra - kvantitativní indikace rozsahu nebo množství určitého atributu, produktu nebo procesu. Počet chyb
Metrika - kvantitativní míra míry do jaké systém, komponenta nebo proces má daný atribut. Počet chyb na 1000 řádků
Charakteristiky metrik založené na velikosti kódu (size oriented)
Velikost softwarového produktu
Lines Of Code (LOC)
1000 Lines Of Code (KLOC)
Vynaložené úsilí v člověkoměsících (E/MM)
Počet chyb/KLOC
Cena/LOC
Počet stránek dokumentace/KLOC

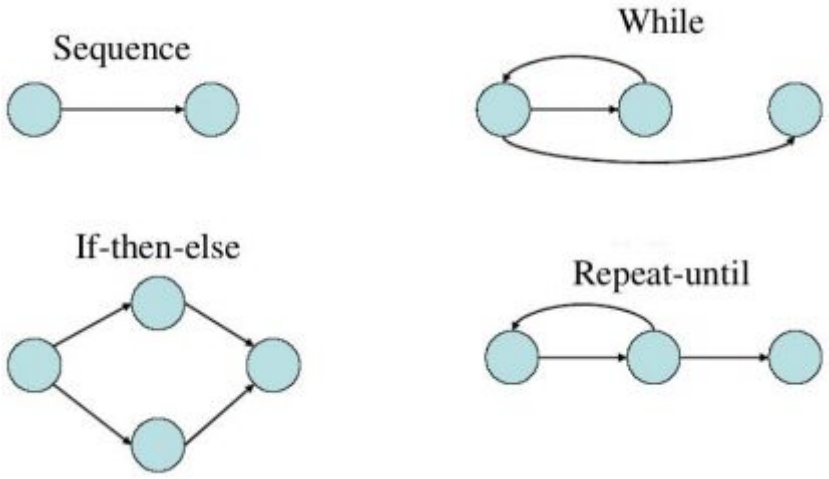
Charakteristiky metrik založené na FP
Chyby na FP, defekty na FP, počet stran dokumentace na FP, počet FP na osobu za měsíc

Halsteadovy metriky

Softwarová metrika výpočtu složitosti na základě statistické analýzy kódu. Používá tyto proměnné:
n1 - počet unikátních operátorů, n2 - počet unikátních operandů, N1 - celkový počet operátorů, N2 - celkový počet operandů
Pomocí nich podle specifických vzorečků definuje tyto metriky:
Length: N = N1 + N2, Vocabulary: n = n1 + n2, Estimated length: Ñ = n1 log2 n1 + n2 log2 n2, Purity ratio: PR = Ñ / N, Volume: V = N log2 n

McCabeovy flow grafy

Jsou založeny na zobrazování kontrolních toků programu. Graf slouží k zobrazení kontrolního toku. Uzly představují úlohy zpracování. Hrany představují tok řízení mezi uzly.



Cyklomatická složitost

Množina nezávislých cest v grafu
V(G) = E – N + 2, E - počet hran grafu, N - počet uzlů grafu
V(G) = P + 1, P - počet predikátových uzlů

Coupling

Datový a regulační tok

d_i - input data parameters, c_i - input control parameters, d_o - output data parameters, c_o - output control parameters

Globální

g_d - global variables for data, g_c - global variables for control

Environmentální

w - fan in number of modules called, r - fan out number modules that call module

Metriky

$M_c = k/m$, $k = 1$
$m = d_i + ac_i + d_o + bc_o + g_d + cg_c + w + r$
a, b, c, k can be adjusted based on actual data

Ukončení projektu

určit, co se dělalo dobře, co bylo špatně, co fungovalo a co ne, jak to udělat příště lépe

Hlavním cílem není „pomoci“ pitvanému projektu, ale organizaci. Analýza je potřebná pro pochopení výkonnosti procesu při řešení daného projektu, ze které získáme poznatky o schopnostech samotného procesu.

Obecné informace a informace vztažené k procesu

Souhrnná informace o projektu, dosažená produktivita a kvalita, použitý proces a jeho odchylky, odhady a aktuální časy, použité nástroje apod.

Rizikové řízení

Odhadnutá rizika včetně plánovaných opatření. Skutečná rizika a jejich vyřešení.

Velikost

Odhady pro jednoduché, středně složité a složité moduly. Sjednocení velikosti podle jedné metriky (např. SLOC ⇒ FP).

Práce

Odhad práce a skutečná práce. Práce zhodnocená podle etap. Cena práce věnované kvalitě (procento práce věnované přezkoumání, testování, přepracování, projektově zaměřené školení).

Defekty

Souhrn nalezených defektů s analýzou významnosti, etap detekce, etap zavedení, efektivita odstranění defektů.

Kauzální analýza

Po ukončení projektu známe výkonnost procesu. Pokud vybočuje výkon z běžných mezí, pak hledáme příčiny. Kauzální analýza zkoumá velké odchylky a identifikuje jejich příčiny, obvykle pomocí diskuse a brainstormingu.

Aktiva procesu a dodatky

Jiné užitečné artefakty procesu, které mohou být potenciálně přínosné pro budoucí projekty.

Unified process

Proces definuje KDO udělá CO, a KDY a JAK to udělá, aby se dosáhl nějaký cíl
Unifikovaný proces je Iterativní a inkrementální, Řízen use casy, Zaměřen na architekturu systému.

Fáze dle UP

- **Zahájení** - Definice rozsahu projektu, vytvoření byznisového cílu
- milník vize
- **Příprava** - Naplánování projektu, specifikace funkcionality, navrhnutí architektury
- milník základ architektury
- **Konstrukce** - Vývoj produktu
- milník počáteční funkcionalita
- **Předávání** - Předání produktu uživatelům
- milník release produktu

Jaké diagramy používá Elaboration fáze Unified Procesu?

- **Diagram tříd** - Popis objektového modelu systému. Znázorňuje objekty, jejich atributy, metody a vztahy mezi ostatními objekty.

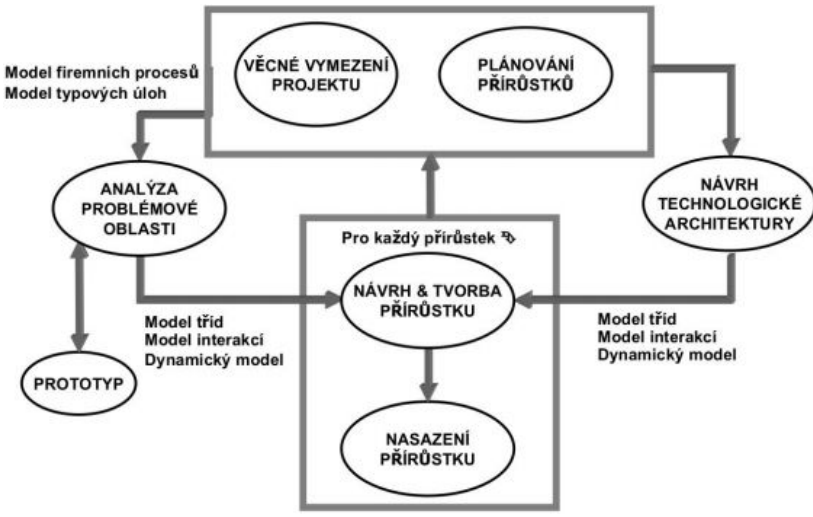
- **Sekvenční diagram** - Sekvenčně popisuje nějaký proces v systému. Znázorňuje komunikaci mezi objekty pomocí zpráv (funkční volání). Objekty "žijí" v průběhu tzv. lifelines.
- **Kolaborační diagram** - Podobný sekvenčnímu diagramu. Sekvence je ale očíslována. Toto číslování specifikuje pořadí volání metod.
- **Stavový diagram** - Znázorňuje takovou událost probíhající v systému, která má konečný počet stavů. Má formu stavového automatu. Jeho součástí jsou stavy, přechody a události.
- **Diagram aktivit** - Zachycuje proměnné chování (flow) nějaké aktivity v systému. Popisuje sekvenci přechodu od jedné aktivity k druhé. Popisuje i paralelní, rozvětvené a souběžné flows systému.

Jaké diagramy se používají v Use Case Model v UP?

- **Use case diagram** - Znázorňuje případy použití systému. Různé role (actors) mohou mít různé případy použití.
- **Sekvenční diagram**
- **Stavový diagram**
- **Diagram aktivit**

Select Perspective

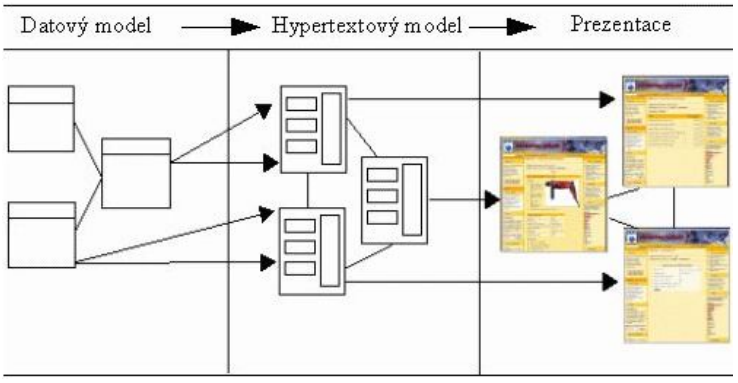
- Je to kombinace procesního modelování a objektově orientované metodiky (Businness Process Reengineering (BPR), Object Modelling Technique (OMT))
- Využívá UML diagramy a procesní mapy
- Inkrementální postup
- Architektura systému založená na komponentách



Popsat modely SP a pořadí jejich vytváření

Modelování firemních procesů, Modelování případů užití, Modelování tříd objektů, Modelování interakcí objektů, Modelování dynamiky objektů, Modelování komponent

WebML (Web Modeling Language)



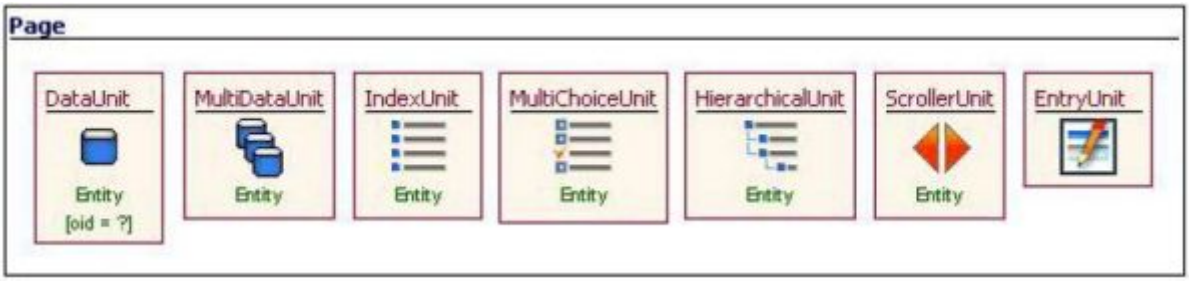
Modelovací nástroj i metodika. Při vývoji (modelování) webů je důležitým prvkem navigace. Umožňuje vytvořit komplexní model webové aplikace. Není „zbytečně" robustní jako např. UML

Podpora v CASE nástrojích - WebRatio Site Development Studio (podporuje generování kódu z navržených modelů), MS Visio (obsahuje šablony diagramů)

Skládá se z těchto modelů - Datový model, Hypertextový model, Prezentační model

Popsat hypertextový model WebML

- Je tvořen dvěma submodely, které jsou spolu těsně spjaty.
- Prvním je **kompozice webové aplikace**, kde je definováno složení prezentace z jednotlivých stránek a složení stránek z jednotlivých předdefinovaných elementů (units). Ty představují atomické součásti obsahu stránky a jsou vázány na entity datového modelu, ze kterých čerpají svůj obsah nebo chování.
- Druhou částí hypertextového modelu je **navigace**, která zobrazuje, jak jsou mezi sebou jednotlivé stránky - respektive jejich elementy provázány. Definuje strukturu a funkčnost webové aplikace na konceptuální úrovni, která je nezávislá na implementačních detailech.



Jaké dva modely používá hypertextový model WebML (a popsat je) - viz nad obrázkem.