Softvérové inžinierstvo

Časť 1: Softvér a proces vývoja softvéru (úvodná prednáška)

Verzia: 1.1, 2.3.2001

Pavol Mederly, pavol.mederly@fmph.uniba.sk

Časť 1: Softvér a proces vývoja softvéru

1.1. Úvod

1.1.1. Jednotlivé druhy softvérových produktov

Softvér sa dnes používa prakticky všade. Priemysel, štátna správa, školstvo, zdravotníctvo – všetky tieto oblasti, a v globále celá ľudská spoločnosť, čoraz viac závisia od schopnosti efektívne vyvíjať a prevádzkovať softvérové systémy s požadovanými vlastnosťami.

Pre získanie základného prehľadu si pripomeňme niektoré v súčasnosti používané druhy softvérových produktov. (Uvedené množiny produktov nie sú disjunktné.)

- **Softvér pre podporu činnosti organizácie** (Business Software): sú to informačné systémy v tradičnom zmysle: ekonomické agendy, riadenie predaja, databáza klientov, evidencia študentov, knižničný softvér, ...
- **Systémový softvér** (System Software): operačné systémy a ich súčasti (napr. drajvre), pomocné programy (editory, kompilátory), špeciálne operačné systémy, napríklad pre komunikačné zariadenia. Príklady: Windows 2000, Linux, Cisco IOS, ...
- **Softvér bežiaci v reálnom čase** (Real-Time Software): programy, ktoré spracúvajú udalosti a riadia procesy vo svojom prostredí. Vyznačujú sa tým, že na udalosti musia reagovať v určenom čase (typicky ide o čas od 1 milisekundy po 1 minútu). Príklady: softvér pre riadenie chemickej továrne, softvér na riadenie fyzikálneho experimentu, ...
- **Softvér ako súčasť iných produktov** (Embedded Software): „Inteligentné produkty“ sa stali súčasťou prakticky každej oblasti priemyslu. Softvér sa používa v chladničkách, mikrovlnných rúrach, automobiloch, lietadlách, mobilných telefónoch, ...
- **Softvér pre vedecké a inžinierske účely** (Scientific and Engineering Software): je charakterizovaný intenzívnymi výpočtami („number crunching“). Aplikácie siahajú od molekulárnej biológie po vulkanológiu, od analýzy zaťaženia prvkov v automobile po skúmanie procesov v raketoplánoch.
- **Softvér pre osobné počítače** (Personal Computer Software): v posledných desiatich rokoch sa trh v tejto oblasti rozšíril obrovským tempom. Kancelársky softvér, spracovanie obrazu, multimédiá, jednoduché databázové systémy, finančné aplikácie, prístup k externým údajom – toto je len niekoľko z množstva aplikácií.
- **Softvér v oblasti umelej inteligencie** (Artificial Intelligence Software): tento softvér sa používa na riešenie problémov, ktoré nie je možné v súčasnosti priamym spôsobom algoritmizovať. Medzi aplikácie patria expertné systémy, systémy pre rozpoznávanie obrazu a hlasu, systémy pre automatické dokazovanie teorém a podobne.

Budeme sa zaoberať softvérom vo všeobecnosti, aj keď v niektorých častiach tejto prednášky sa zameriame na informačné systémy (Business Software).

1.1.2. Generické produkty a produkty na objednávku

Softvérové produkty spadajú do dvoch širokých tried:

1. Generické produkty (majúce stovky, tisíce, stotisíce používateľov)
2. Produkty vytvárané na objednávku pre konkrétneho zadávateľa

Metódy, ktoré sa pri vývoji používajú, sú pre tieto dve triedy podobné. Výrazným rozdielom je, že pri prvom druhu produktov sú požiadavky na produkt formulované na základe údajov marketingového oddelenia vývojovej organizácie, v druhom prípade sú dané zadávateľom.

V tejto prednáške budeme implicitne uvažovať o produktoch na objednávku. Avšak prakticky všetky princípy, ktorými sa budeme zaoberať, sa dajú použiť aj pre vývoj generických produktov.

1.1.3. Produktivita v oblasti vývoja softvéru

Softvérová kríza (pojem, ktorý sa objavil na konferencii NATO venovanej softvérovému inžinierstvu v r.1968) – neschopnosť dostatočne kvalitne a efektívne vyvíjať softvér – tento pojem sa objavil v súvislosti s príchodom tretej generácie počítačov, kde možnosti hardvéru prudko prevýšili možnosti vývoja adekvátneho softvéru.

Problémom je to, že napriek existujúcim poznatkom v oblasti softvérového inžinierstva sa pri vývoji softvéru často postupuje nesystematicky, čo má za následok zlyhania a neúspechy pri realizácii mnohých projektov. Podľa údajov z literatúry má približne polovica vyvíjaných produktov problémy s prekročením plánovaného rozpočtu. Čo sa týka použiteľnosti výsledného produktu,

- v 5% prípadov je produkt dodaný a používaný tak, ako uvádza špecifikácia,
- v 55% je produkt dodaný a používaný po modifikáciách,
- v 10% je produkt síce dodaný, ale v praxi nepoužívaný,
- v 15% je produkt používaný po jeho úplnom prepracovaní,
- v 15% prípadov nie je produkt vôbec dodaný (projekt končí neúspechom).

Ak by sme mali zhrnúť hlavné problémy pri vývoji softvéru, situácia vyzerá takto:

- prekračovanie rozpočtu a plánovaného termínu dodávky,
- nesplnenie používateľských požiadaviek,
- neuspokojivá kvalita produktu.

Pod kvalitou softvéru rozumieme jeho vlastnosti:

- správnosť – správať sa podľa špecifikácie
- robustnosť – slušne sa chovať v prípadoch mimo špecifikácie
- flexibilnosť – cez jednoduchosť návrhu a slabo zviazané moduly
- znovupoužiteľnosť – ovplyvňuje aj iné faktory (napr. spoľahlivosť)
- kompatibilnosť (jednoduchosť, ako sa systém spája s okolím)

Tiež ďalšie vlastnosti:

- Efektívnosť
- Portabilnosť – jednoduchosť prenesenia na inú platformu
- Verifikovateľnosť – jednoduchosť overenia správnosti softvéru
- Integrita – ochrana pred neoprávneným prístupom a modifikáciou
- Jednoduchosť používania

Nie všetky faktory kvality sa dajú dosahovať naraz. (integrita vs. jednoduchosť používania, efektívnosť vs. portabilita/flexibilnosť/znovupoužiteľnosť).

Brooks konštatuje, že neexistuje „čarovný prútik“, ktorý spraví vývoj softvéru jednoduchou záležitosťou (pod takýmto prútikom si niektorí predstavujú rôzne metódy a techniky, objektovo-orientovaný prístup, znovupoužiteľnosť, patterny, CASE, RAD, vizuálne nástroje, ...). Všetko sú to dôležité veci, ktoré pomáhajú, ale samy osebe nepredstavujú „čarovný prútik“.

Napriek tomu však možno pozorovať každoročný nárast produktivity v tejto oblasti približne o 6%, čo predstavuje zdvojnásobenie produktivity približne každých 12 rokov a zodpovedá situácii v ostatných oblastiach priemyslu.

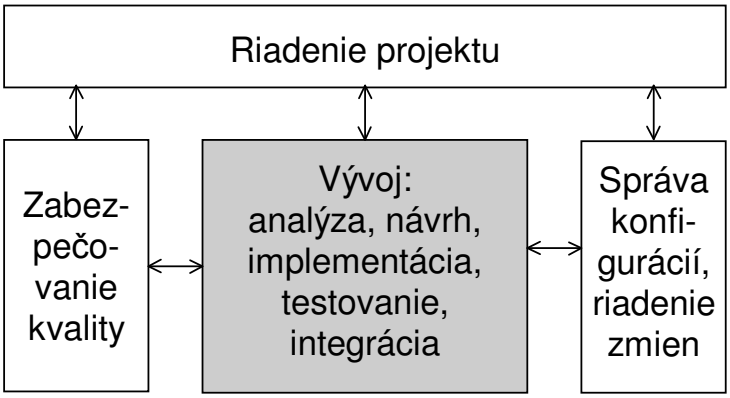
Softvérové inžinierstvo sa zaoberá teóriou, metódami a nástrojmi potrebnými pre vývoj softvéru.

1.2. Proces vývoja softvéru (Software Process)

1.2.1. Štruktúra procesu vývoja

Proces vývoja softvéru je množina aktivít, ktorých konečným výsledkom je softvérový produkt. V každej organizácii je tento proces iný. V každej „píšu softvér“ trochu iným spôsobom. Rozdiely môžu byť napríklad v intenzite dokumentovania alebo v intenzite testovania.

V každom prípade však vývoj softvéru prechádza vždy viac-menej rovnakými etapami: analýza a špecifikácia požiadaviek, návrh, implementácia a testovanie, integrácia, údržba. Vývoj je podporovaný aktivitami, ako sú: riadenie projektu, zabezpečovanie kvality, správa konfigurácií a riadenie zmien.



1.2.2. Zadávateľ, riešiteľ, používateľ

Prv, než prejdeme k modelom procesu vývoja softvéru, uveďme tri základné úlohy pri vývoji softvéru: zadávateľ, riešiteľ a používateľ.

- **Zadávateľom** (klientom) je organizácia (prípadne jednotlivec), ktorá má záujem o vývoj softvérového produktu.
- **Riešiteľmi** (vývojovými pracovníkmi) nazývame pracovníkov zodpovedných za vývoj produktu. Môže ísť o pracovníkov organizácie klienta alebo o pracovníkov externej dodávateľskej organizácie.
- **Používatelia** sú osoby, ktoré budú softvérový produkt priamo používať.

Zadávateľom môže byť napríklad vedúci ekonomického oddelenia niektorej spoločnosti a riešiteľom môže byť tím pracovníkov oddelenia informatiky tejto spoločnosti. Používateľmi môžu byť v tomto prípade pracovníci ekonomického oddelenia.

Iný príklad: zadávateľom je ministerstvo obrany, dodávateľom niektorá z veľkých softvérových spoločností zaoberajúcich sa vývojom softvéru pre armádne účely.

Samozrejme, vývoj môže mať omnoho menšie rozmery: zadávateľom a zároveň používateľom môže byť advokát v súkromnej praxi, riešiteľom študent, ktorý si privyrába vývojom softvéru.

Ak sú zadávateľ a riešitelia súčasťou jednej organizácie, situácia je v istom zmysle jednoduchšia: ak ide o rôzne organizácie, musí medzi nimi existovať zmluva presne popisujúca produkt, ktorý má byť vytvorený.

Úlohy môžu byť aj ďalšie, napr. prevádzkovateľ systému.

1.2.3. Stručný popis jednotlivých etáp procesu vývoja softvéru

Počas svojho života prechádza softvérový produkt približne nasledujúcimi etapami.

Etapa analýzy a špecifikácie požiadaviek

Etapa analýzy a špecifikácie požiadaviek predstavuje transformáciu neformálnych požiadaviek zadávateľa do komplexného a štruktúrovaného popisu požadovaného produktu.

Na prvom stretnutí medzi zadávateľom a riešiteľmi zadávateľ načrtne produkt, tak, ako si ho predstavuje. Vo všeobecnosti zadávateľ nie je odborník v oblasti softvérového inžinierstva, a tak jeho predstavy o produkte môžu byť z pohľadu riešiteľov vágne, vnútorne nekonzistentné alebo jednoducho nerealizovateľné. Úlohou riešiteľov je presne zistiť, čo zadávateľ požaduje a zistiť, aké sú jeho obmedzenia na proces vývoja. Typickými obmedzeniami sú náklady a termín („výsledný produkt musí stáť do 8 000 000 Sk a musí byť hotový do 14 mesiacov“), avšak časté sú aj požiadavky týkajúce sa spoľahlivosti produktu („musí byť funkčný 99% času“), jeho veľkosti („musí bežať na mojom osobnom počítači“) a podobne.

Na ďalších stretnutiach medzi zadávateľom a riešiteľmi sú požiadavky na produkt upresňované a analyzované z pohľadu realizovateľnosti a z pohľadu ekonomickej účelnosti.

Môžu tu nastať problémy. Všeobecným problémom je, že zadávateľ si jednoducho *nevie predstaviť*, ako by mohol vyzerať softvér, ktorý požaduje. Nevie teda ani formulovať svoje požiadavky. Iným problémom je, že *nevie, čo potrebuje*. Napríklad ak má firma zaoberajúca sa predajom finančné problémy a požiada o systém pre správu financií zameriavajúci sa na veci, ako predaj, mzdy, účtovníctvo, toto nemusí pomôcť, ak skutočným problémom sú krádeže v obchodoch a v skladoch.

Riešitelia na základe informácií získaných od zadávateľa vypracúvajú podrobný a presný popis požiadaviek na výsledný produkt. Používajú pritom prostriedky prirodzeného jazyka, rôzne diagramy a prípadne aj formálne metódy.

Je nevyhnutné, aby bol tento popis požiadaviek jednoznačný, úplný a vnútorne konzistentný.

Príklad nejednoznačnosti: „Z databázy sa načíta identifikátor súčiastky nasledovaný identifikátorom kontajnera. Ak tento obsahuje písmená AQ, treba cenu prenesenia danej súčiastky do daného kontajnera násobiť koeficientom 1,5.“ (Ktorý identifikátor má obsahovať písmená AQ ?)

Príklad neúplnosti: napríklad ak špecifikácia nepopisuje, čo sa má spraviť, ak sú na vstupe nekorektné údaje.

Príklad vnútornej nekonzistentnosti: Na jednom mieste je uvedené: „Ak tlak v pracovnej nádobe presiahne 10 atmosfér, je potrebné okamžite uzavrieť ventil M17.“ Na inom mieste:

„Ak tlak v pracovnej nádobe presiahne 10 atmosfér, je potrebné upovedomiť operátora. Ak operátor nezareaguje do 30 sekúnd, treba uzavrieť ventil M17.“

Na túto etapu sa môžeme pozerať ako na 2 súvisiace pod-etapy: analýzu požiadaviek (požiadavky sa získavajú a analytik sa v nich snaží vyznať) a špecifikáciu požiadaviek (analytik vytvára výsledný dokument s popisom požiadaviek). V praxi sú však tieto činnosti úzko spojené.

Etapa návrhu

V predchádzajúcej etape sme sa zaoberali tým, *čo* má produkt robiť. Cieľom etapy návrhu je určiť, *ako* bude produkt dané požiadavky realizovať.

Vychádzajúc zo špecifikácie požiadaviek riešitelia navrhnú vnútornú štruktúru produktu, t.j. jeho rozdelenie na moduly. Stanovia vzájomné väzby medzi modulmi a celkový spôsob riadenia v systéme (napríklad centralizovaný vs. založený na udalostiach). Funkciou modulu môže byť autentifikácia používateľa alebo zobrazenie katalógu produktov (v systéme pre elektronické obchodovanie). Modul v riadiacom systéme lietadla môže dostať postupnosť súradníc prichádzajúcej strely, spočítať jej trajektóriu a upozorniť pilota na možné ohrozenie.

Rozdelenie na moduly a popis ich vzťahov predstavuje prvú časť návrhu, tzv. *architektonický návrh*.

Počas dekompozície na moduly je potrebné zaznamenať všetky rozhodnutia, ktoré boli spravené. Jednak kvôli prípadnému opravovaniu návrhu a tiež kvôli budúcemu rozširovaniu produktu počas údržby, kedy je často potrebné pôvodný návrh modifikovať.

Po architektonickom návrhu nasleduje *podrobný návrh*. Tu sa podrobne popíšu jednotlivé moduly, navrhnú sa pre ne detaily používateľského rozhrania, dátové štruktúry a príslušné algoritmy.

Porovnanie medzi „analýzou“ a návrhom – analýza a špecifikácia požiadaviek pracuje v „priestore problému“ (problem space), návrh pracuje v „priestore riešenia“ (solution space).

Etapa implementácie a testovania modulov

Cieľom tejto etapy je programovo realizovať jednotlivé moduly a vypracovať podrobnú dokumentáciu k vytvoreným programom.

V tejto etape je vykonávané aj *testovanie* jednotlivých modulov. Ide jednak o neformálne testovanie samotným programátorom a tiež o formálne testovanie vzhľadom k vopred určeným testovacím vstupom. Pre zabezpečenie správnosti programov sa používa tiež technika *revízie kódu* (code review), kedy programátor vedie členov revízneho tímu cez vytvorený kód.

Etapa integrácie a testovania systému

V tejto etape prebieha spájanie modulov a testovanie produktu ako celku. Spôsob spájania modulov (spojenie všetkých naraz alebo postupné pridávanie; pridávanie zdola nahor alebo zhora nadol) môže mať významný vplyv na kvalitu výsledného produktu.

Počas integrácie sa vykonáva *integračné testovanie*, kedy sa testujú spojenia medzi modulmi.

Po ukončení integrácie sa produkt *testuje ako celok*. Testuje sa jeho funkčnosť podľa špecifikácie požiadaviek a testujú sa tiež ďalšie požiadavky, napríklad doba odozvy, robustnosť. Skontroluje sa, či je zdrojový kód a dokumentácia kompletná.

Na záver sa vykonáva *akceptačné testovanie*. Toto vykonáva zadávateľ, a to v prostredí, kde bude produkt prevádzkovaný a na svojich vlastných údajoch.

Etapa prevádzky a údržby systému

Od chvíle, keď je produkt akceptovaný zadávateľom, predstavujú všetky ďalšie zmeny v ňom

jeho *údržbu*. Údržba je integrálnou súčasťou procesu vývoja softvéru. Predstavuje typicky 1/2 až 2/3 nákladov vynaložených na vývoj produktu počas jeho života a je potrebné na ňu myslieť od začiatku. Napríklad v etape návrhu treba brať do úvahy, pokiaľ je to možné, prípadné budúce rozširovanie systému. Podobne pri písaní kódu programov je potrebné myslieť na ich budúcu údržbu.

Častým problémom pri údržbe je nedostatočná, nekvalitná, resp. neaktuálna dokumentácia. Mnohokrát jedinou dokumentáciou k produktu je jeho zdrojový kód.

1.2.4. Modely procesu vývoja softvéru

Proces vývoja softvéru je potrebné riadiť. Z tohto vyplýva požiadavka jednotlivé aktivity vývoja istým spôsobom usporiadať, štruktúrovať.

Schéma, podľa ktorej proces vývoja postupuje, sa nazýva *modelom procesu vývoja softvéru* (Software Process Model). Musíme si uvedomiť, že skutočný proces vývoja býva zložitejší ako jeho schématické znázornenie modelom procesu vývoja.

Neexistuje „dobrý“ a „zlý“ model procesu vývoja softvéru. Každý z modelov softvérového procesu je vhodnejší pre niektoré triedy aplikácií. Pre iné môže znamenať nižšiu kvalitu alebo použiteľnosť výsledného produktu.

Proces vývoja softvéru má svoje atribúty. Uveďme niektoré z nich:

- **pochopiteľnosť** (understandability) - do akej miery je proces explicitne definovaný a ako ľahké je jeho definíciu pochopiť ?
- **viditeľnosť** (visibility) - prinášajú jednotlivé aktivity procesu jasné výsledky a umožňujú tak pozorovať postup práce ? Toto je dôležitý prvok pre to, aby bol proces manažovateľný, vzhľadom na to, že softvér ako taký je neviditeľný (na rozdiel - napríklad - od stavby budovy, kde sa stav prác dá skontrolovať podstatne jednoduchšie).
- **možnosť podpory** (supportability) - do akej miery je možné jednotlivé aktivity podporiť prostriedkami CASE ?
- **rýchlosť** (rapidity) - ako rýchlo môže prebehnúť proces dodania výsledného produktu na základe danej špecifikácie ?

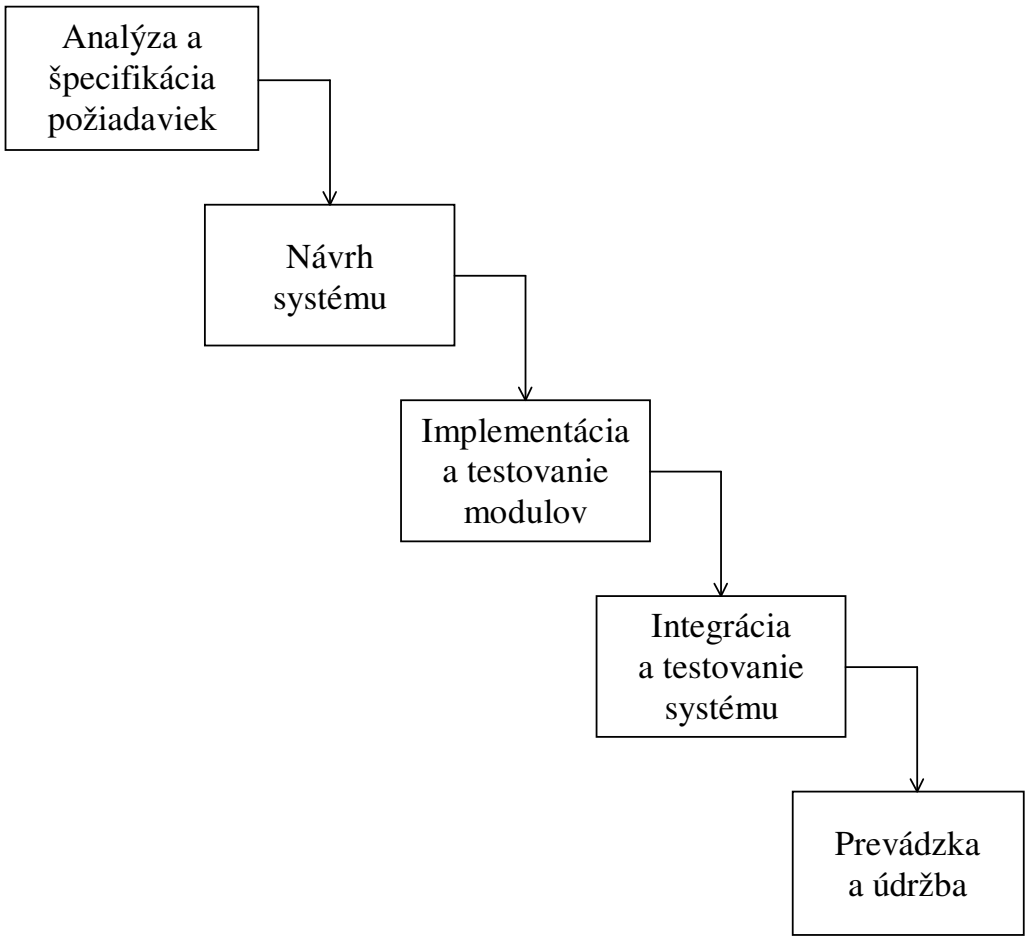
Nie je možné optimalizovať všetky tieto atribúty naraz. Príkladom čiastočne si protirečiacich atribútov sú viditeľnosť a rýchlosť. Ak má byť proces rýchly, môže byť nevyhnutné znížiť jeho viditeľnosť, pretože robenie procesu viditeľným vyžaduje častejšie vytváranie dokumentov, čo proces nevyhnutne spomaľuje.

1.2.4.1. Vodopádový model procesu vývoja softvéru

Charakteristickou črtou vodopádového modelu je, že vývoj systému je rozdelený do sekvenčne usporiadaných etáp zodpovedajúcich jednotlivým vývojovým aktivitám (analýza a špecifikácia požiadaviek, návrh, implementácia a testovanie, integrácia).

Práce na danej etape môžu začať až vtedy, keď je predchádzajúca etapa ukončená a všetky jej produkty sú zdokumentované a schválené.

Návrat k niektorej z predchádzajúcich etáp je možný. Často sa stáva, že napríklad počas návrhu sa nájdu nedostatky v špecifikácii požiadaviek alebo počas implementácie sa nájdu nedostatky v návrhu, či dokonca v špecifikácii. Vtedy sa vývoj vráti do príslušnej etapy, vykonajú sa zodpovedajúce úpravy, inovované dokumenty prejdú schválením a pokračuje sa vo vývoji.



Historicky ide o prvý explicitný model procesu vývoja softvéru (rok 1970). Bol odvodený z iných inžinierskych procesov a ako prvý poskytol možnosť spraviť proces vývoja relatívne viditeľným.

Existuje mnoho variantov tohto modelu, ktoré sa líšia prakticky len rozdelením na etapy.

Vodopádový model má mnohé prednosti aj nedostatky.

Prednosti vodopádového modelu:

1. Vodopádový model zavádza do vývoja softvéru disciplínu: vývoj musí postupovať systematicky od jednej etapy k ďalšej.
2. Vodopádový model požaduje, aby výstupom každej etapy bola úplná dokumentácia, čo je veľkým prínosom najmä pre budúcu údržbu produktu.
3. Tým, že je proces rozdelený na etapy a v každej etape sa vytvárajú definované dokumenty, je proces vývoja viditeľný.
4. To, že pri tomto procese vývoja sú známe (resp. mali by byť známe) všetky požiadavky pred tým, ako sa začína návrh systému, a tiež to, že je vytvorený podrobný návrh pred tým, ako sa začína implementácia, sú predpoklady pre to, aby bolo pri použití vhodných metód možné vytvoriť dobre štruktúrované programy.

Nedostatky vodopádového modelu:

1. Ak riešitelia na začiatku pochopia požiadavky zadávateľa nesprávne (napríklad aj vďaka

tomu, že ich tento nie je schopný jasne formulovať), táto skutočnosť často vyjde najavo až pri akceptačnom testovaní a vtedy bude náprava veľmi nákladná. Riziko nesprávneho pochopenia požiadaviek je väčšie, ak je špecifikácia požiadaviek založená len na dokumentoch. (T.j. ak sa nepoužijú žiadne prostriedky na vytvorenie prototypov systému.)

2. Je prirodzené, že požiadavky na produkt sa časom menia, či už v dôsledku zmien vonkajšieho prostredia, alebo v dôsledku lepšieho uvedomenia si problematiky a možností produktu zadávateľom. Vodopádový model dokáže tieto zmeny realizovať len ťažko (návratom k etape analýzy a špecifikácie požiadaviek a pokračovaním vývoja od tejto etapy).
3. Ak nebude môcť byť vývoj z nejakého dôvodu (napr. finančného) dokončený, nemá zadávateľ k dispozícii žiadny použiteľný softvérový produkt.
4. Zadávateľ môže mať problémy s nasadením celého produktu naraz.

1.2.4.2. Inkrementálny model procesu vývoja softvéru

Podstatou inkrementálneho modelu je, že vývoj produktu prebieha po jednotlivých *inkrementoch*. Inkrement predstavuje istú časť celkovej funkčnosti produktu. (Teda nejde nutne o skupinu modulov – modul v systéme môže obsahovať časti kódu patriace rôznym inkrementom.)

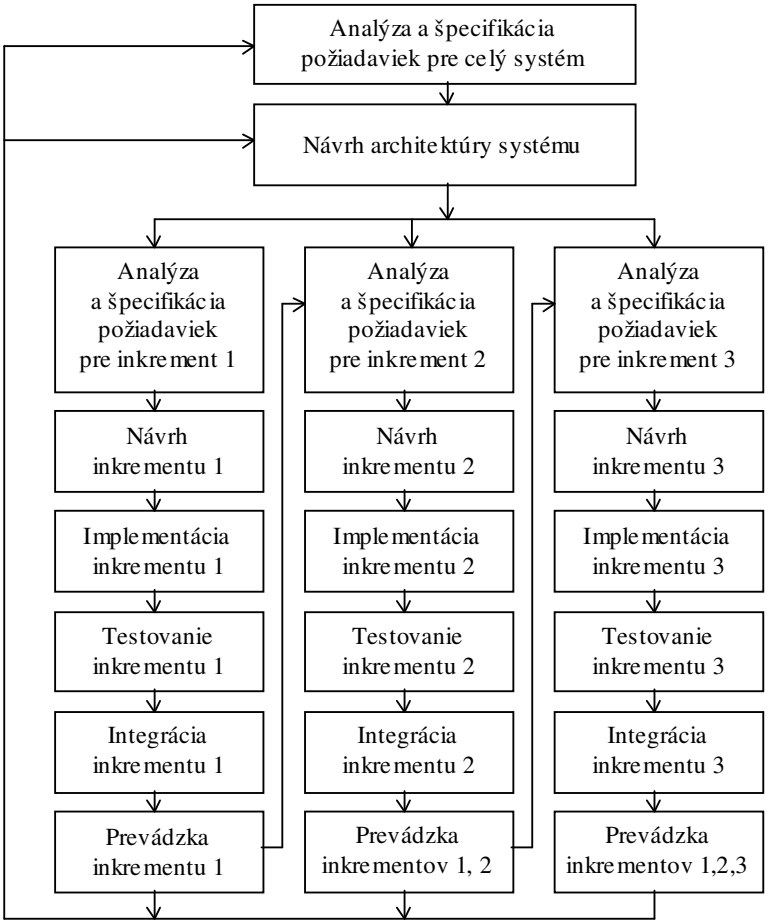
Napríklad ak je úlohou produktu predávať knihy cez Internet, jedným z inkrementov môže byť práca s katalógom kníh (zobrazenie, vyhľadávanie atď). Ďalším registrácia používateľov. Iným zase objednávanie kníh.

V operačnom systéme môže byť jedným inkrementom scheduler, ďalším systém pre správu súborov.

Pri vývoji sa najskôr analyzujú a popíšu požiadavky na produkt ako celok a navrhne sa architektúra produktu. Potom sa postupne realizujú jednotlivé inkrementy. Pre každý z nich prebehne detailná analýza a špecifikácia požiadaviek, návrh, implementácia, testovanie a integrácia (jednak integrácia modulov v danom inkremente a tiež integrácia inkrementu do zvyšku produktu). Vždy po takejto iterácii sa produkt môže odovzdať zadávateľovi do (čiastočnej) prevádzky.

Vývoj končí vtedy, keď sa dosiahne celková požadovaná funkčnosť produktu.

Riešiteľ môže rozdeliť produkt na inkrementy ľubovoľným spôsobom, s podmienkou, že po zakomponovaní každého z inkrementov musí byť (medzi)produkt ako celok testovateľný. Typicky môže ísť o 10 až 50 inkrementov.



Porovnanie vodopádového a inkrementálneho modelu

Cieľom vodopádového modelu je dodať zadávateľovi prevádzkovateľný produkt, pokrývajúci všetky zadávateľove požiadavky. Pri použití inkrementálneho modelu dostáva zadávateľ v jednotlivých iteráciách prevádzkovateľný produkt, avšak taký, ktorý pokrýva len podmnožinu jeho požiadaviek. Výsledná funkčnosť je dosiahnutá až v poslednej iterácii.

Prednosti inkrementálneho modelu:

1. Pri použití vodopádového modelu dostáva zadávateľ produkt až na konci vývoja, ktorý môže trvať mesiace až roky. Naproti tomu pri inkrementálnom modeli dostáva prevádzkyschopnú časť produktu omnoho skôr, niekedy už po niekoľkých týždňoch.
2. Skúsenosti s prevádzkou už nasadenej časti produktu môžu zadávateľovi pomôcť lepšie špecifikovať požiadavky na zvyšok produktu.
3. Keďže produkt (resp. jeho časť) je v prevádzke skôr ako pri vodopádovom modeli, návratnosť vynaložených investícií je vo všeobecnosti lepšia.
4. Zmeny v požiadavkách, ktoré sa vyskytnú, sa pri inkrementálnom modeli do produktu ľahšie zakomponúvajú.
5. Zadávateľ môže produkt nasadzovať po častiach, čím sa zavádzanie uľahčí.
6. Ak sa vývoj z nejakých dôvodov neukončí, zadávateľ má k dispozícii funkčný produkt spĺňajúci aspoň časť z jeho požiadaviek.

Nedostatky inkrementálneho modelu:

1. Systém musí mať otvorenú architektúru – tak, aby nebolo pri pridávaní jednotlivých inkrementov potrebné robiť veľké zmeny v už vytvorenom produkte. Vytvorenie takejto architektúry je náročnejšie, na druhej strane je však táto neoceniteľná pri ďalšom rozširovaní systému počas jeho údržby.
2. Inkrementálny model je v porovnaní s vodopádovým modelom náročnejší na vývoj aj na riadenie.

1.2.5. Prototypovanie

Prototyp je program, ktorý je funkčne ekvivalentný časti výsledného produktu.

Napríklad ak má výsledný produkt spracovávať účtovníctvo a evidenciu pre sklad, prototyp by mohol byť program, ktorý umožní vkladať príslušné údaje a tlačiť správy, avšak nebude kontrolovať správnosť vložených údajov, zabezpečovať ukladanie údajov do databázy a podobne.

Prototyp pre produkt, ktorý má určovať koncentráciu enzýmu v roztoku by mohol byť program, ktorý načíta údaje, spočíta a zobrazí výsledok, avšak nebude podporovať kontrolu vstupných údajov, ani grafické zobrazovanie výsledkov.

Takéto prototypy sa používajú na zisťovanie a overovanie požiadaviek zadávateľa.

Postup:

- Na základe predbežných požiadaviek zadávateľa sa vytvorí prvý prototyp pre budúci produkt. Počas tvorby tohto prototypu sa aktivity analýzy, návrhu, implementácie a testovania prekrývajú. Na kvalite kódu nezáleží, ide len o to, aby bol prototyp vytvorený čo najrýchlejšie.
- Zadávateľ a budúci používatelia budú skúšať vytvorený prototyp a na základe práce s ním budú upresňovať svoje požiadavky.
- Riešitelia budú na základe aktualizovaných požiadaviek prototyp upravovať, a to dovtedy, kým nebudú zadávateľ a budúci používatelia s prototypom spokojní.

Na základe schváleného prototypu sa štandardným spôsobom (vrátane spísania požiadaviek) vytvorí požadovaný produkt. Kód prototypu sa vo všeobecnosti vo výslednom produkte **nemá** použiť.

Poznámka: Uviedli sme prototypovanie ako techniku na zisťovanie, resp. overovanie požiadaviek. Prototypy sa vytvárajú aj v ďalších etapách vývoja, napríklad na overenie realizovateľnosti predkladaného návrhu systému.

Prototypovanie nepredstavuje model procesu vývoja softvéru. Je to technika, ktorú je možné použiť pri ľubovoľnom z modelov.

1.3. Literatúra

Stephen R. Schach: **Software Engineering**, 2nd ed., Aksen Associates, 1993

Ian Sommerville: **Software Engineering**, 5th ed., Addison-Wesley, 1996

Roger S. Pressman: **Software Engineering: A Practitioner's Approach**, McGraw-Hill, 1997