Luboš Brim **Softwarové inženýrství**

sepsal Jiří Dobeš

20. dubna 1995

Obsah

1 Úvod

Znaky reálného softwaru (máme na mysli velké projekty, řádově statisíce řádků kódu) jsou

- velikost
- pro více (velmi mnoho) počítačů a uživatelů
- více pracovníku ve vývoji
- na začátku vývoje není podrobná specifikace
- modifikace (údržba)
- vazby na prostředí (vývoj HW je rychlejší, nemá smysl měnit SW při každé změně HW)

PROBLÉMY

- náklady - překračují se (2 a více-krát)
- termíny - prodlužují se
- spolehlivost, kvalita
- údržba
- přenositelnost

Programování v malém $\not\Longrightarrow$ Programování ve velkém.

FAKTA

- cena HW klesá, cena SW roste
- možnosti HW se zvětšují, to si žádá nový SW (12% ročně)
- nových programátorů přibývá asi 4% ročně
- je potřeba více údržby a změn stávajícího SW než nových aplikací.

$\diamond$

PROBLÉMY + FAKTA $\Longrightarrow$ SW krize (od 70.let)

$\diamond$

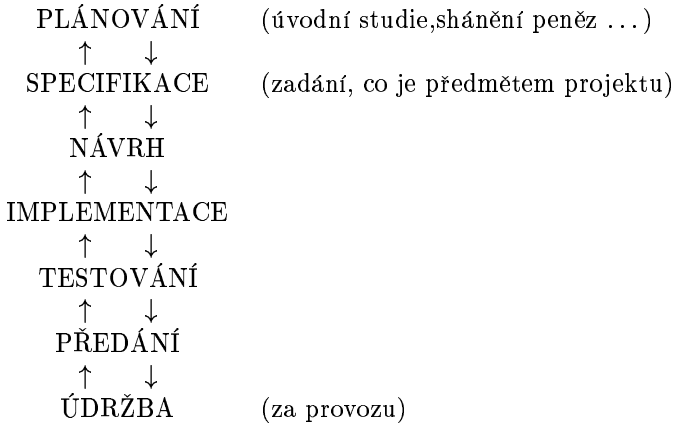
Jako řešení se objevuje **SOFTWAROVÉ INŽENÝRSTVÍ**, které prosazuje tyto zásady:

- Tvorba software jako inženýrského produktu.
- Rozdělení vývoje systému do etap.

- Standardní techniky, prostředky a metody.
- Řemeslná dovednost.

Softwarové inženýrství je vlastně taková praxe programování (nutná vysoká produktivita práce programátora). Snahou je aplikovat inženýrské metody práce při tvorbě složitých informačních systémů za účelem dosažení větší efektivity, spolehlivosti a rychlosti vývoje.

2 Etapy vývoje softwaru



V procesu vývoje je nutno se vracet, řízení a kontrola vyžadují management. Fázím návrhu, implementace a testování se také říká **projektování**.

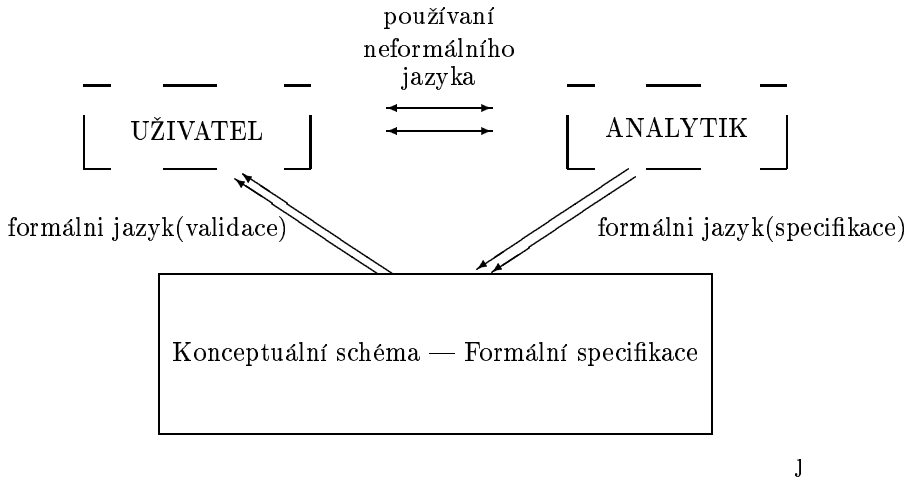
2.1 Specifikace

- má charakter smlouvy mezi výrobcem a odběratelem.
- je základem pro vývoj.
- je východiskem pro ověření, zda hotový produkt odpovídá specifikaci.
- vymezuje požadavky — funkční nebo nefunkční (tj. omezení).
- je základem pro kontrukci prototypu — nefunkční požadavky a nedůležité funkční požadavky zanedbáme, náklady na prototyp jsou menší, současně získáme představu o nákladech na celý vývoj. Formální specifikace → automatické prototypování.
- důležitá úloha uživatele — uživatel má zodpovědnost za navržení špatné specifikace

2.2 Implementace

V každé etapě máme určitou reprezentaci systému:

etapa	reprezentace
specifikace	přirozený jazyk
	formální specifikační jazyk
návrh	
	pseudokód
implementace	
	programovací jazyk



Obrázek 1: Spolupráce uživatele a analytika při tvorbě specifikace

2.3 Počítačový systém

1. Analýza a definice systému z hlediska funkce, výkonnosti a kvality
2. Altenativní řešení z hlediska:
   - ekonomické stránky
   - technické analýzy
   - pracovních sil
   - prostředí nasazení

Z analýzy a alternativních řešení se vybere jedna konfigurace.
Systém je rozdělen do tří částí :

- Hardwarový
- Softwarový
- Ostatní, např. obsluha.

Vznikne popis konfigurace a definice systému (včetně rozdělení funkcí mezi HW,SW a ostatní). Toto se nazývá **úvodní studie.**

2.4 Etapy vývoje SW podsystému

- Předprojektová příprava (plánování)
- Projektování (výroba)
- Nasazení do užívání a údržba

Na různých úrovních se ve vývoji objevují změny. Existují dva možné přístupy ke změnám:

1. nepřipustí se změny.
2. naopak změny se připouští, při návrhu již s touto možností počítáme; je nutná dobrá struktura systému.

2.4.1 Předprojektová příprava

navazuje na úvodní studii.

Stanovení cílů projektu — co, proč se má dělat (základní funkce, záměr), s čím (jaké technické prostředky jsou k dispozici, pracovníci), cena, doba

Z těchtou projekčních záměrů vznikne **projektový úkol**.

Schválení z ekonomického hlediska — není-li, hledají se alternativy.

Zpracování **technického projektu**
Stanovíme (vymezíme) :

- funkce
- vazby (mezi prvky systému)
- omezení (na rychlost, dat, spolehlivost,...)
- kriteria úspěšnosti (vymezení části funkcí, vlastností systému, vzhledem ke kterým zadavatel pozná, zda mu vyhovuje; Chceme, aby toto bylo součástí zadání.)

2.4.2 Projektování

je tvořeno dvěmi fázemi:

- **Předběžný návrh** — návrh struktury (architektury) systému.
- **Podrobný návrh** — návrh vlastních algoritmů modulů. V této etapě vzniká **prováděcí projekt**.
- Kódování — algoritmy mohou být popsány prostředkem různým od cílového jazyka . Zde je do cílového jazyka převedeme (transformujeme).
- Testování — ve fázi návrhu algoritmu se musí prověřovat správnost, nepoužívá se testování, ale verifikace (invarianty, ...), i když jen pro klíčové funkce. Testování nemůže potvrdit správnost, může pouze ukázat, že program je špatně. *Systematicky ukazujeme, že program nefunguje.* Jsou zde fáze:testování modulů, vazeb, celého SW podsystému.

2.4.3 Nasazení do užívání a údržba

Údržba — odstraňování objevených chyb, dopracování změn, přenos na jiný počítač atd.

3 Plánování

3.1 Analýza a definice systému

Cílem definice systému je:

(a) Vyjasnění pojmů
(b) Popis funkce; vazby, výkonnost
(c) Předběžný návrh struktury systému
(d) Alokace funkcí HW,SW a dalším prvkům
(e) Odhad nákladů a termínů
(f) Návrh plánu vývoje

3.2 Analýza problému

1. Plán analýzy — rozmyslet co s kým, jak budeme dělat, koho z odborníků budeme potřebovat, kolik to bude stát.
2. Kontakty s uživateli — s kým, kdy budu mluvit, struktura organizace, jaké mají zkušenosti s počítač. systémem, jaký systém u nich už chodí.
3. Cíle systému — účel systému, proč se systém dělá, jaký bude mít vliv.
4. Poznání současného systému — může být i nepočítačový.
5. Datové prvky a struktury —(není nutné) používaná data, formuláře, výkazy ...
6. Výzkum jiných systémů — co podobného je na trhu, cena, rozsah ...
7. Alternativní řešení — nejde o to navrhovat zde alternativní řešení, ale kriteria, která mají splňovat.
8. Analýza struktury systému.
9. Plánování dalších etap.

3.3 Rozbor proveditelnosti

dává odpověď na otázku, zda lze udělat a za kolik (v rámci omezení). Omezení:

- Ekonomická proveditelnost : přínos systému × náklady, doba návratnosti.
- Technická proveditelnost : analýza technických omezení, technologie atd.
- Alternativy : neuvažujeme pouze jedno řešení, alternativy srovnáváme vzhledem k těmto omezením.

DOKUMENTACE K ROZBORU PROVEDITELNOSTI

1. Úvod — shrnutí, o jaký projekt jde.
2. Charakterizuje účel systému, prostředí.
3. Doporučení — pro další vývoj.
4. Alternativní řešení — popsány alternativy, formulována kriteria porovnávání alternativ a výběru doporučovaného řešení.
5. Popis systému — odkaz na úvodní studii (její pasáže), vymezuje systém.
6. Analýza nákladů a přínosů — ekonomické zdůvodnění systému.
7. Technická analýza.

3.4 Náklady a přínosy

Přínosy:

Přímý lze vyjádřit v Kč nebo jiné měně.

Nepřímý tvoří ostatní, obvykle větší část.

Přínosy informačního systému z hlediska řízení:

- Sníží se náklady na výpočty a tisky
- Možnost rychlé změny hodnot, parametrů,...
- Úplnější a systematičtější vytváření dat

- Zvýšení kapacity dat
- Standardizace postupů
- Zvýšení bezpečnosti při práci s daty
- Zvýšení přenositelnosti
- Rychlejší přístup k informacím

Náklady:

- Pořízení systému — nákup,výroba, instalace
- Změny v prostředí provozu (přestavba, propuštění lidí)
- Změny související s přizpůsobováním systému novému programu (např. nová verze OS)
- Komunikační prostředky (telefony, linky)
- Zaškolení uživatelů
- Náklady na prvotní pořízení dat (již existujících)
- Údržba
- Elektrická energie

Taktéž je třeba myslet na budoucnost — rozšíření systému.

3.5 Alokace funkcí

Alternativy:

(a) Všechny funkce realizovat softwarově.
(b) Některé funkce (jednoduché) realizovat hardwarově,ostatní softwarově.
(c) Vše hardwarově.

Je třeba zhodnotit alternativy z těchto hledisek:

- Cena
- Počet vytvořených (prodaných) systémů
- Výkonnost (HW řešení umožňuje větší rychlost, ale je dražší)
- Oprava vzniklých chyb (u SW je to jednodušší a levnější)
- Spolehlivost

4 Specifikace

ÚVODNÍ STUDIE

Úvod : cíl, účel systému, prostředí v němž bude nasazen, rozsah vývoje, proveditelnost a výkonnost, přehled o nákladech a termínech

Popis funkcí systému : popis jednotlivých funkcí, které má systém plnit

Alokace funkcí : navržena konfigurace realizace funkcí — HW,SW,obsluha atd.

Omezení : finanční, technická, bezpečnostní, ...

Náklady : lze jen odhadem

Termíny realizace : také odhadem

Dokument se píše obvykle ve formě odborného článku.

4.1 Schvalování

Používají se tato hlediska:

- Řídící a organizační (manažerské) — hodnotí se z hlediska organizačního , zda byla vymezena rizika, zda odpovídají ceny, budoucí trendy, ...
- Technické — zda jsou v systému všechny funkce, dostupné technologie, ...

◇

Do této chvíle jsme se zabývali celým systémem (HW + SW + ostatní). Dále se zaměříme již jen na softwarový podsystém.

◇

4.2 Stanovení cílů a účelu SW podsystému

Jinak řečeno plánování softwaru. Cílem je zjemnit informace z definice systému pro vývoj SW, abychom odhadli lépe náklady, plánování, ...

Zpřesnění popisu systému:

- Funkce — dekomponujeme funkci na řadu podfunkcí.
- Definice vazby — rozhraní, předávané hodnoty, synchronizace.
- Výkonnost — omezení na dobu zpracování, paměť atd. Obvykle souvisí s funkčností (zjemnění funkcí — lepší čitelnost — menší výkonnost a naopak).
- Spolehlivost.

Vazby (rozhraní) jednoho systému na druhý. V této fázi vnitřní vazby SW podsystému nevyhodnocujeme. Zabýváme se vazbami softwaru na hardware a na ostatní části systému. Součástí budoucího projektu je také nějaký již existující SW (OS,DBS,...).

Spolehlivost — je obtížné kvantitativními prostředky stanovit její míru.

4.3 Prostředky

Prostředky jsou v zásadě dvojího typu: HW+SW a lidské síly. Míra jejich nasazení se v době vývoje mění.

Pracovníci:

- analytik
- programátor
- právník
- specialisté (na sítě, testování, HW)

- manažer

Potřeba těchto lidí je v čase různá (ale např. programátora potřebujeme i při analýze, je to prostě variabilní).

Počet pracovníků — závisí na velikosti projektu, je těžké odhadnout

Hardware —

- vývojový
- cílový
- jiné HW-ové prvky

Často je vývojový a cílový HW totožný, ale není to podmínkou.

Software —

- podpůrný — slouží pro vývoj (editor,překladač,prostředky podporující jednotlivé etapy vývoje), je třeba zvážit jeho přínosnost
- pomocný — existující,je součástí vyvíjeného systému (databázový jazyk)

4.4 Odhady nákladů

Zajímají nás náklady z hlediska plánování potřeby pracovníků, doby vývoje, potřeby prostředků. Základem jsou charakteristiky produktivity práce — zde je základní problém: jak charakterizovat.

Měření produktivity programátorů: počet odladěných řádků (člověk a měsíc — *čm*). U ostatních profesí ji měřit prakticky nelze (zvláště analytici). V praxi lze nejlépe měřit produktivitu u celé skupiny dohromady (bývá asi 5–7 řádků na den). Z toho je možno odhadnout potřebné počty pracovníků. Dnes existuje tzv. softwarová fyzika, která dává metody pro odhady.

Odhadujeme tyto základní veličiny: *počet instrukcí* nebo *úsilí (v čm)*. Tyto hodnoty bohužel nebývají jednoznačné. U systému nás zajímají tyto charakteristiky:

- Náklady
- Termíny
- Velikost dokumentace
- Počet pracovníků
- Úsilí nebo velikost — toto jsou hodnoty, z kterých se předchozí odvozují.

Odhad počtu instrukcí navrhujeme pesimistický, optimistický a očekávaný. Odhad úsilí děláme zvlášť na analytické, programátorské práce, testování, ... Často provádíme nezávisle oba odhady. Jejich velký rozdíl nám ukazuje na chybu v některém odhadu.

Produktivitu práce ovlivňuje:

- Vývojové prostředí.
- Řešený problém — při řešení již dříve zvládnutého problému je produktivita větší.
- Spolehlivost systému.

Důvody problémů při odhadu

- Techniky odhadů využívají historická data (o dřívějších projektech).
- Nejdůležitějším faktorem je kvalita řešitelského týmu, ale tyto týmy se mění. Mění se i obecné podmínky vývoje (HW,SW,...). Proto je téměř každý systém unikátem.
- Podmínky (zadání i obecné) se mění i během vývoje.

INFORMACE O PROJEKTECH, které bychom měli schraňovat, abychom na jejich základě mohli provádět odhady nových:

1. Identifikace projektu
   - popis hlavních funkcí
   - počet řádků
   - relativní složitost
   - vynaložené úsilí
   - doba vývoje
   - počet pracovníků
2. Cena projektu
3. Cena hlavních funkcí
4. Dokumentace (její velikost)
5. Použité prostředky
6. Záznam o údržbě

Rozložení úsilí do jednotlivých etap:

40 % — 20 % — 40 %

analýza a definice — vývoj — testování

Dojde-li ke **zpoždění**, pak řešením může být:

- Přidání lidí (⇒ zvětšení zpoždění —*Brookův zákon*)
- Vylepšení komunikace.

Nejlépe je se zpožděním počítat už při odhadu (nasadit více lidí).

TÝMY — organizace skupiny pracovníků

1. Každý pracovník dostane několik úkolů tak, aby mezi sebou co nejméně komunikovali. Případná komunikace se děje přes vedoucího.
2. Skupině několika pracovníků se přidělí úkol.
3. Vytvoříme týmy, každý má určitou strukturu (vedoucí,...)

Obecně praxe dává při rozsáhlejších projektech přednost týmům.

Organizace týmu hlavního programátora :

- hlavní programátor (vedoucí) — řídí koordinaci v týmu, také se podílí na pracech.
- zástupce — musí být připraven na to, aby nahradil či zastoupil vedoucího
- programátoři-analytici (2–5)

Velikost týmu je asi 5–10 lidí. Navíc:

+ specialisté
+ pomocný personál
+ knihovník

Nejsou trvalou součástí týmu. Schaňují dokumentaci, zálohují, hlídají standardy podniku, vytváří data o produktivitě.

PROJEKTOVÝ ÚKOL (Dokumentace vzniklá při plánování) :

1. Záměr
   (a) Úvod (co,proč)
   (b) Hlavní funkce
   (c) Jiné charakteristiky (spolehlivost)
   (d) Postup vývoje
2. Prostředky
   (a) Pracovníci
   (b) HW
   (c) SW
   (d) Nasazení prostředků
3. Náklady
4. Termíny

4.5 Technický projekt systému

1. Specifikace, přesné vymezení toho, co má systém dělat. Na základě této specifikace se bude projekt vytvářet. Je zde důležitá *účast uživatele.*
2. Je třeba stanovit:
   - tok a strukturu informací (vymezení datové struktury + tok)
   - funkce systému (zjemnění) — funkční požadavky
   - vazby
   - omezení — nefunkční požadavky

   Opět na tomto základě navrhneme alternativy. Součástí dokumentu budou kriteria úspěšnosti (zjednodušená). Na základě těchto kriterií si potom uživatel systém přebere, i když my budeme systém vyrábět podle kriterií určených ve specifikaci.
3. Předběžný uživatelský manuál — prvotní podoba, jak se systémem zacházet. Tento manuál zastupuje uživatele ve fázi vývoje (jeho pohled na systém), nesouvisí s funkcí systému, ale ovládáním.

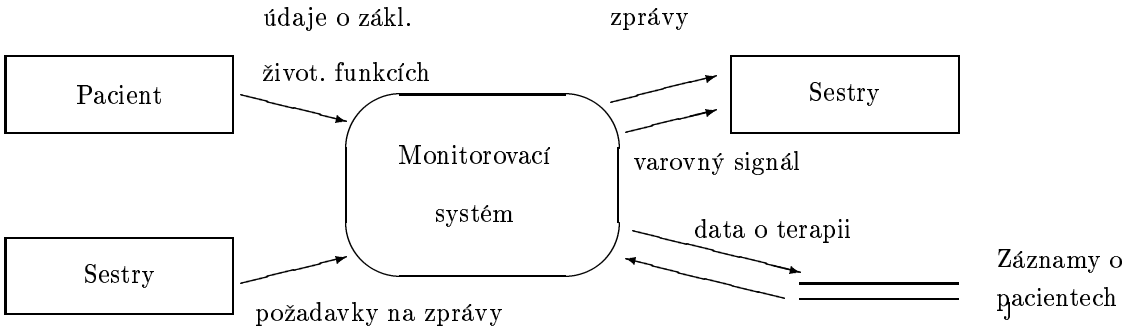
TOK A STRUKTURA INFORMACE Informace má dvě charakteristiky, které spolu velice úzce souvisí:

→ Tok — co vstupuje (def.obor), co vystupuje (obor hodnot), jak je to měněno (funkce).

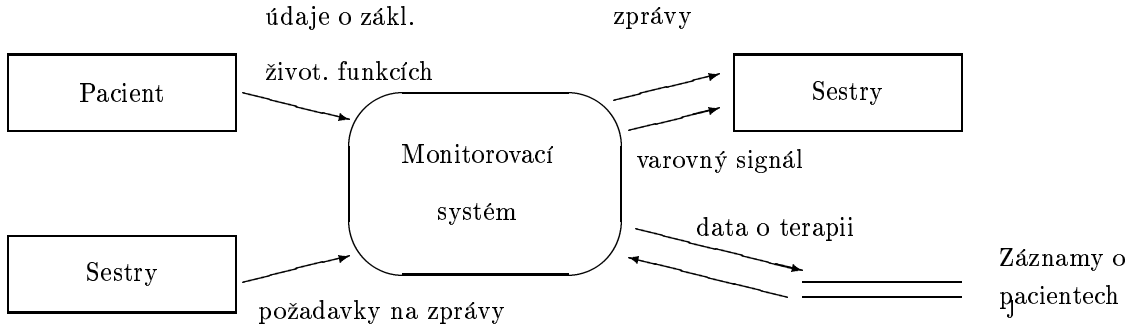
→ Struktura — komponenty, složení.

Jednoduchý obrázek by se měl objevit již na začátku specifikace. Je vždy lépe srozumitelný než specifikační jazyk, nehledě na to, že bude snadněji pochopitelný pro uživatele. Nevýhodou obrázků je jejich omezená velikost. Při větším počtu objektů a vazeb se graf stává nepřehledným a nikdo nepřesvědčí uživatele, aby luštil obrázek, kde je dvacet obdélníčků a desítky šipek mezi nimi. Řešením je rozložit popis systému do hierarchicky členěných obrázků.

Tok informace můžeme reprezentovat pomocí **GTD — grafu toku dat** (běžně používaný anglický termín a zkratka jsou Data Flow Diagram, DFD).
Graf obsahuje tyto položky:



Př. Jednotka intenzivní péče JIP.



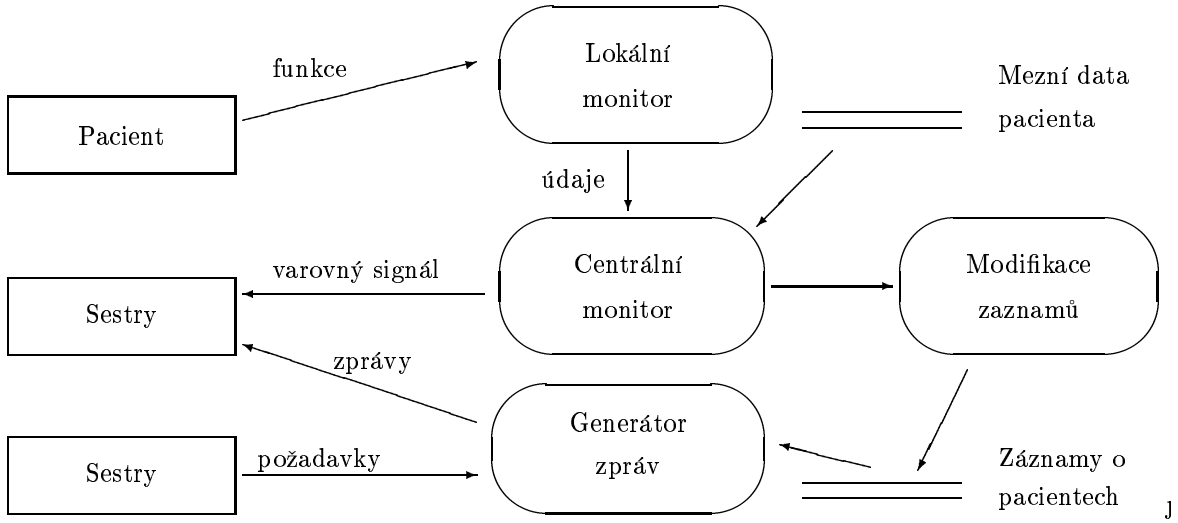
Obrázek 2: Příklad grafu toku dat

Krok zjemnění (viz obrázek3) reprezentuje naše rozhodnutí, že systém bude obsahovat:

1. lokální monitor (u lůžka)
2. centrální monitor
3. generátor zpráv
4. doplňování a modifikace údajů

Zásady vytváření GTD:

- Je to graf toku dat (ne řízení).
- Rozumná velikost (doporučuje se do 8 objektů na obrázku).

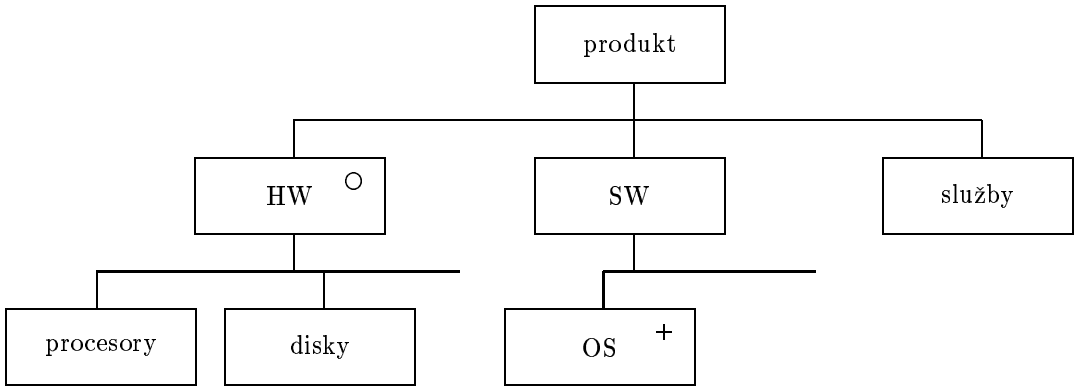


Obrázek 3: Příklad zjemnění grafu toku dat

- Vše označovat.
- Zjemňujeme v každém kroku pouze jeden uzel — v procesu zjemnění musí být zachována spojitost informace.

Struktura informace = datové struktury. V systémech hromadného zpracování dat (HZD) se často setkáme s hierarchickými (stromovými) strukturami dat. Stromová struktura realizuje hierarchii informace.

Př. Hierarchický diagram (Warnierrovy diagramy), lze zajistit opakování kategorie i podmíněný výskyt kategorie. produkty firmy IBM:



Obrázek 4: Příklad hierarchického diagramu

TECHNICKÝ PROJEKT SYSTÉMU

1. Úvod — cíle, záměr
2. Popis zpracovávání informace
   (a) GTD
   (b) reprezentace struktury dat

   (c) vazby informací — informační model
   (d) popis vnějších systémových vazeb
   (e) popis vnitřních vazeb

3. Popis funkcí
   (a) funkce — neprocedurálně, nealgoritmicky
   (b) způsob zpracování — nepříliš detailně
   (c) omezení — def.obor, rychlost zpracování

4. Kriteria úspěšnosti
   (a) výkonnost
   (b) očekávané chování
   (c) třídy testů
   (d) speciální požadavky

Součástí dokumentu je předběžný uživatelský manuál.
Technický projekt se musí zrevidovat, schválit, zhodnotit, případně vrátit. Stává se součástí kontraktu.

CASE prostředky slouží k ulehčení práce, usnadňují technickou práci, například kreslení GTD. Nesou s sebou nebezpečí, že o problému méně přemýšlíme.

5 Projektování systému

Fáze:

1. Předběžný návrh (architektura systému) — způsob, jakým budou spolu jednotlivé moduly komunikovat, nemusí odpovídat GTD.
2. Podrobný návrh (procedury) — návrh algoritmů pro moduly, může být použit i jiný jazyk než cílový.
3. Kódování (implementace)
4. Testování — proces hledání chyb, ukazování nesprávností.

Během projektování vzniká dokumentace (po částech).

PROVÁDĚCÍ PROJEKT

1. Záměr
   (a) Cíle projektu
   (b) Vazby v systému (HW,SW,ostatní)
   (c) Hlavní SW funkce
   (d) Hlavní omezení

2. Dokumentace (průběžně doplňovaná)
   (a) Softwarová dokumentace
   (b) Systémová dokumentace
   (c) Technická dokumentace

3. Popis návrhu (vzniká v rámci předběžného návrhu)
   (a) Popis dat

   i. Přehled toku dat
   ii. Přehled struktury dat
   (b) Architektura systému
   (c) Vazby uvnitř struktury
4. Moduly — pro každý modul :
   (a) Popis způsobu práce
   (b) Popis vazeb
   (c) Popis modulu (v jazyce návrhu)
   (d) Použití modulu
   (e) Organizace dat
5. Struktura souborů a globálních dat
   (a) Externí soubory
      i. Logická struktura
      ii. Popis logických záznamů
      iii. Přístupové metody
   (b) Globální data
   (c) Vzájemná reference souborů a dat
6. Vzájemné reference na požadavky
7. Testování
   (a) Způsob testování modulů
   (b) Integrace (sestavování)
   (c) Speciální požadavky
8. Předběžný instalační manuál a operační manuál

Další body jsou možné.

6 Návrh

6.1 Kontroly návrhu

— se vyplatí.

— provádějí se častěji nejen na konci.

— lze změřit, co nás stojí, ale obtížně určíme, co nám přinesou.

Kontroly můžeme přibližně rozlišit na *formální* a *neformální*. Cílem kontrol je odhalit chyby, nejasnosti, nedostatky,...

Formální kontroly slouží k závěrečnému zhodnocení, provádí se po dokončení větších celků.

- Kontrola vyžaduje přípravu ze strany projektanta i uživatele.
- Velký počet účastníků.
- Formální řízení kontroly — vyžaduje naplánování.
- Výsledkem jsou zprávy o kontrole.

Nicméně neformální kontroly jsou účinnější.

Plánování kontroly

- Dva týdny předem zaslat účastníkům nutnou dokumentaci.
- Vedoucí kontroly vyžaduje předem písemné připomínky.

Cílem je vyhodnotit, zkontrolovat produkt (nikoli projektanta — osobní pohnutky !) a odhalit chyby, ne už jejich oprava a řešení.

Neformální kontroly (např. strukturovaná revize) jsou

- vyvolány tím, že se projekt vytváří v týmu (týmech).
- mezi formálními kontrolami.
- často na úrovni předběžného návrhu (specifikace je obtížnější než algoritmizace).
- plánovány, ale ne v tak velkém rozsahu jako formální kontroly.
- efektivní.
- nejsou náhradou formální kontroly.

Forma:

- Příprava
- Vlastní kontrola
- Opravy
- Následná revize

Účastníci: (malý počet — asi 5, mohou být kumulované funkce)

- Autor
- Koordinátor (moderátor)
- Zapisovatel
- Oponenti

Aktivity účastníků:

- Autor připraví dokument, navrhne termín a oponenty.
- Koordinátor zorganizuje průběh celé akce.
- Oponenti se musí s dokumentem předem seznámit, připravit se na oponenturu, musí najít chyby, připomínky dá písemně autorovi.
- Zapisovatel musí problematice rozumět, zapisuje to podstatné, musí přesně vystihnout nedostatky.

Vlastní kontrola proběhne poměrně krátce — asi 30–45 minut. Velikost dokumentu musí odpovídat délce kontroly, aby se dal zvládnout.

Závěry kontroly:

- Akceptováno — nedostatky nejsou příliš závažné.
- Akceptováno s opravami — nedostatky jsou formulované v zápisu.
- Je nutno svolat následnou kontrolu.

Doporučení:

- Cílem je chyby odhalit, ne opravit.
- Dělat časově krátké kontroly.
- Nedělat více než 2 kontroly po sobě.
- Kontrolovat logické celky.
- Nutnost respektovat koordinátora.

6.2 Kvalita softwaru

Evidentní rozlišení: Fungující × Nefungující program.
Jemněji:

- Z hlediska struktury.
- Z hlediska procedurálních aspektů.

Struktura — Software tvoříme jako model reality. Ta je popisována komponentami systému, jejichž propojení (struktura systému) je základním znakem vyjadřujícím složitost systému.

Procedurální aspekty — Jde o použití kvalitních algoritmů, jejich systematický návrh, zvyšování efektivity algoritmů atd.

Několik pojmů:

Modulárnost softwaru

- je předpokladem zvládnutí složitých SW systémů
- je na úkor efektivnosti a zvyšuje složitost systému
- některé systémy musí být monolitické (real-time,OS), ale mohou být modulární alespoň v logickém smyslu.

Princip utajení informace

- použit v OOP a jazyku Ada
- je výhodný pro snižování chybných interakcí mezi moduly systému

Nezávislost modulů

- spojení — míra propojení s okolím
- soudržnost — počet funkcí, které modul sdružuje — tyto dva pojmy jsou v protikladu.

Složitost systému je dvojího druhu: strukturální × procedurální.

6.3 Návrh struktury na základě toku dat

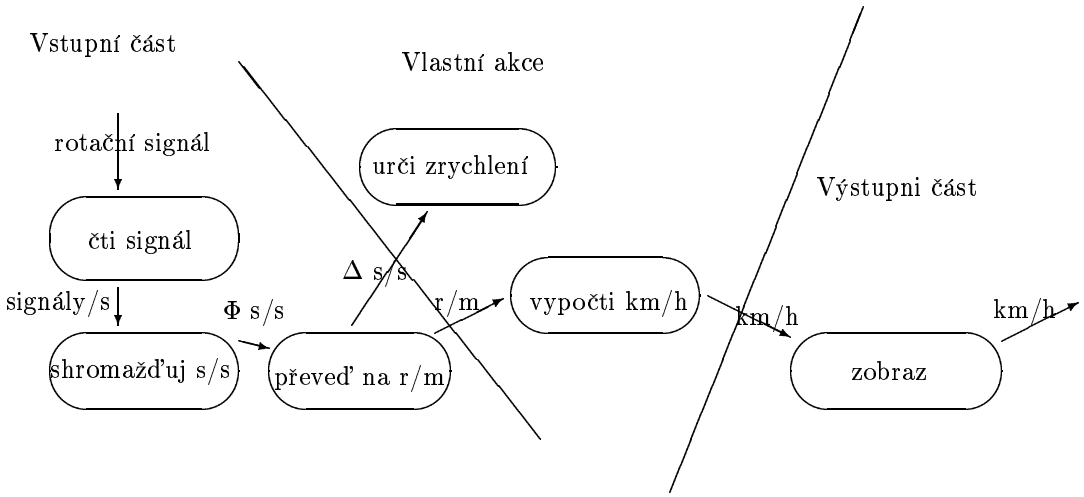
Postup při návrhu:

1. Ustanovujeme kategorie toku dat
2. Vymezíme hraniční oblasti
3. GTD transformujeme na strukturu SW
4. Provedeme vylepšení

Kategorie toku dat jsou dvojí:

- transformace — tři fáze: příprava vstupních dat, vlastní transformace a úprava do výstupního formátu
- transakce — má jen fáze vstupu a zpracování informace, výsledkem je nějaká činnost ovlivňující systém (analogie katalyzátoru)

Příklad: Mikropočítačový A/D převodník, který na povel zobrazuje údaje, indikace rychlosti, otáčky, spotřeba paliva, cena na 1km,... Navíc indikátor překročení rychlosti a vybrané funkce.



Obrázek 5: Příklad grafu toku dat: A/D převodník pro měření rychlosti

6.4 Návrh struktury na základě struktury dat

Vhodná je hierarchická struktura dat. Postup při návrhu:

1. Vyhodnocení struktury dat
2. Data reprezentujeme pomocí elementárních struktur (sekvence, výběr, opakování)
3. Struktura dat je zobrazena do řídící struktury SW
4. Vylepšena
5. Nakonec procedurální popis

Zde bohužel není zcela jasný přechod mezi předběžným a podrobným návrhem.

Jacksonova metoda návrhu je založena na použití hierarchických diagramů pro reprezentaci dat. Hierarchický diagram má opět podobu stromu, jehož kořenem je popisovaná datová struktura, uzly na nižší úrovni představují složky. Východiskem pro návrh diagramu jsou diagramy struktury vstupních a výstupních dat a korespondence mezi nimi. Postupnou úpravou a přizpůsobováním obou diagramů dostaneme po doplnění podmínkami a operacemi kostru struktury programu.

6.4.1 Principy návrhu dat

1. systematický přístup
2. datové typy — jaké operace a funkce s těmito daty budeme provádět
3. volba programovacího jazyka

6.5 Prostředky pro podrobný návrh

Grafické prostředky — vývojové diagramy, strukturogramy, blokové diagramy. Jejich výhodou (?) je podpora strukturovaného programování. Ve výuce programování to může být vhodný nástroj, ale v praxi příliš omezující a svazující.

Tabulární prostředky — tzv. rozhodovací tabulky jsou speciálním nástrojem sloužícím k znázornění složitých rozhodovacích podmínek do přehledných tabulek.

Jazykové prostředky
- programovací jazyky — nevýhodné, pro přílišnou podrobnost syntaxe, používané pro návrh algoritmů (nekompiluje se !!); GCL od Dijkstry (je například nedeterministický)
- pseudokódy — obsahují řídící struktury a možnosti návrhu (abstraktních) datových typů

Co by tyto prostředky měly umožňovat:

- modulárnost — návrh modulů a vazeb mezi nimi
- strukturovanost
- jednoduchost — pro snadnost používání
- produkt je strojově zpracovatelný
- automatické provádění
- kódování — přepis do programovacího jazyka

6.6 Programovací jazyky a programování

Volba programovacího (cílového) jazyka :

- Jsou situace, kdy nevolíme, protože zákazník požaduje konkrétní jazyk
- Podle aplikace — DBS, expertní systém,...
- Dostupnost — je, není, za kolik
- Prostředí nasazení
- Podle znalosti programátorského týmu

6.7 Kódování

KOMENTÁŘE kódu jsou také součástí dokumentace.

Úvodní komentáře (modulu) by měly obsahovat:

1. Funkce modulu
2. Popis rozhraní — parametry, příklad volání, volané podprogramy
3. Globální idea algoritmu (pokud nějaká existuje)
4. Důležitá lokální data
5. Autor, verze, testováno,...

Komentování kódu nechť dodržuje tyto zásady:

1. Výstižná jména identifikátorů
2. Formátování textu kódu i komentářů odsazováním
3. Stručnost, přesnost, výstižnost
4. Standardizace deklarací dat, komentovat strukturovaná data
5. Nekomentovat 1 řádek (není-li to nutné), ale logický úsek
6. Jeden příkaz na řádku, více příkazů jen zřídka, souvisí-li nějak spolu
7. Srozumitelnost pro jiné
8. Nepoužívat nadstandardní rysy jazyka.

VSTUPY/VÝSTUPY převažovaly dříve dávkové (uživatel zadá vše, pak se začnou vykonávat příslušné akce). Dnes se používají interaktivní způsoby práce, které jsou pro uživatele výhodnější, ale o to hůř se programují.

Body, které by měl interaktivní V/V program, či aspoň ta jeho část, která se vstupem a výstupem zabývá, mít:

- komunikace musí být odolná vůči uživateli (blbuvzdorná)
- potvrzování důležitých údajů a rozhodnutí
- pomoc uživateli při zadávání dat (default hodnoty,)
- rozlišovat více úrovní uživatelů (začátečník, pokročilý, atd.) při bohatosti nápověd
- časové omezení doby vstupů (timeout)
- možnost návratu do předchozích stavů
- minimalizace práce uživatele při chybě

EFEKTIVITA systému je přímo úměrná kvalitě návrhu. Jen jednou větou: systém navrhujeme tak efektivní, jak je požadováno, nikdy ne více.

7 Testování a kvalita systému

Testování je stejně důležité jako programování. Zabírá až 40 % celkového úsilí, u některých systémů i více (bezpečné systémy). Má destruktivní charakter — úspěšný test je takový, který odhalí chyby.

Testovací konfigurace zahrnuje plán testování, testovací data, očekávané výsledky testování a kritéria úspěšnosti. *Odhady spolehlivosti* jsou u řídících systémů nutné.

Neodhalí-li testování chyby, můžeme předpokládat, že je chybné !

Testování:

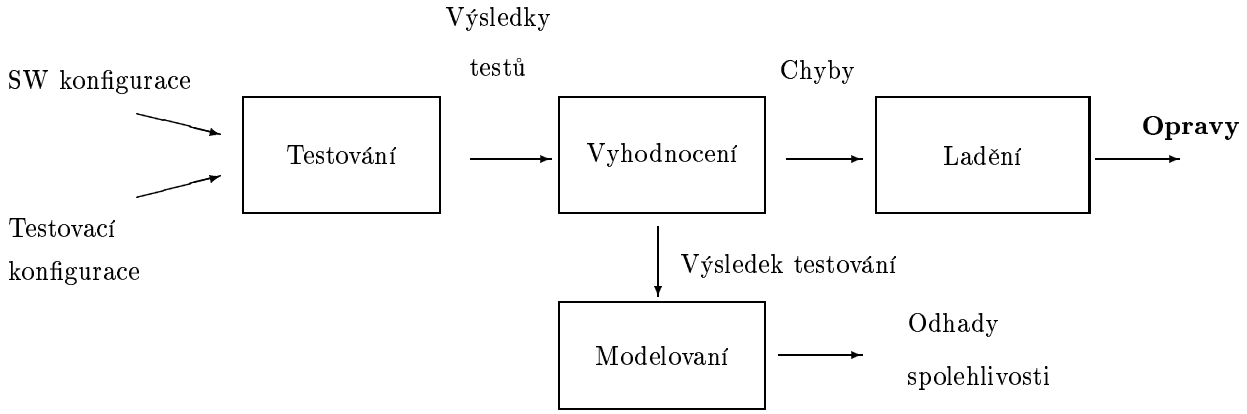
- má dvě základní formy. Systém jako :

  Černá skříňka : systém chápeme jako celek, provádíme test odezvy systému na určitý vstup.

  Bílá skříňka : zajímá nás činnost a správná funkce vnitřních částí. Nezajímá nás, co systém dělá, ale jak to dělá. Samozřejmě je zde nutné testovat části systému selektivně, nelze všechny.

- je jednou z nejdůležitějších metod, které ovlivňují kvalitu výsledného systému.
- by měla provádět samostatná vývojová skupina, což je výhodnější z hlediska specializace i efektivnosti, členy této skupiny by však měli být i vývojáři. Tato skupina by měla být přítomna od počátku vývoje.

Postup při testování:



Obrázek 6: Postup při testování

1. Testování modulů — vzhledem k podrobnému návrhu.
2. Integrované testování (testy na vazby mezi moduly) — vzhledem k předběžnému návrhu.
3. Validace (test SW podsystému jako celku) — vzhledem ke specifikaci, technickému projektu.
4. Testování systému (HW + SW + ostatní; často formou zkušebního provozu) — vzhledem k úvodní studii. Zkušební provoz (je-li možný), je levnější než plné testování.

SPECIFIKACE TESTOVÁNÍ

1. Záměr testování — základní funkce a vlastnosti, na které se zaměříme při testování.
2. Plán testování :
   - Navržení globální strategie, jak rozdělit testování do částí podle logického členění systému — **fáze**
   - Příprava testovacích dat, kriterií a očekávaných dat
   - Časový rozvrh
   - Pomocný software (spouštěcí moduly)
   - Pomocné prostředky — zvláštní SW,HW (například pro testování real-time systémů modely řízených celků; nebo CASE prostředky pro podporu testování)
3. Postup testování — pro každou fázi je třeba určit:
   - Testované moduly
   - Pořadí integrace
   - Pomocné prostředky
   - Testovací data
   - Očekávané výsledky
4. Skutečné výsledky

7.1 Testování modulů

má tyto charakteristické vlastnosti:

- Testy probíhají formou bílé skříňky.
- Můžeme testovat více modulů současně.

- Zaměření se na důležité struktury systému:
  - ⋆ rozhraní
  - ⋆ lokální datové struktury
  - ⋆ důležité cesty
  - ⋆ zpracování chyb
  - ⋆ hraniční situace
- Před testováním musíme připravit testovací data a očekávané výsledky, testovací program pro volání testovaného modulu a „prázdné“ podprogramy z modulu volané. Byla-li by výroba těchto programů příliš nákladná, testujeme celek až v následující fázi.

7.2 Integrované testování

nazýváme také integrace, protože zde vlastně sestavujeme celý projekt z částí, které byly dosud vyvíjeny samostatně.

- Testujeme vazby mezi moduly
- Sestavujeme systém z modulů do celku
- Integrace je možná v zásadě dvěma způsoby:
  - ⋆ zdola nahoru (postupné připojování)
  - ⋆ shora dolů
    - ◦ do hloubky — jdeme po určité důležité větvi (častější).
    - ◦ do šířky — přidáváme moduly po skupinách.

Shora dolů je metoda používaná obecně jako základní.

- Hlavní řídící modul je použit jako testovací, podřízené jsou nahrazeny prázdnými, ty jsou postupně nahrazovány skutečnými, podle toho, jak se při testování proniká do hloubky.
- Přidáváme vždy jen jeden modul.
- Provádíme regresní testování — opakované provádění dřívějších testů.
- Základní výhoda — od počátku máme „celý“ systém.
- V případě, kdy je nutno testovat vazbu, která vyžaduje správnou funkci podstatně nižších částí, můžeme testování této části odložit nebo, což je typické, kombinujeme s postupem zdola nahoru.

Zdola nahoru je integrování po shlucích stejně významných nebo funkčně příbuzných modulů.

- Výhoda je, že není třeba pomocných modulů, jen jednoduché spouštěcí programy pro volání testovaných modulů.

Nejrozumnější je kombinace obou metod podle potřeb a možností, které máme.

7.3 Validace

- prověřuje systém vzhledem ke kriteriím úspěšnosti.
- testuje systém jako *černou skříňku.*
- kontroluje konfiguraci systému.
- najde-li chybu, je nutno:
  - ⋆ hledat společně se zákazníkem řešení (složitá jednání !!), protože chybu nelze v daném termínu (odevzdání projektu) již opravit.

  - ⋆ se dohodnout na posunutí termínu odevzdání systému.
  - ⋆ rychle určit časovou náročnost opravy (příčina chyby).
- je-li nalezena odchylka, ne chyba, možná ji uživatel akceptuje (nějaká drobnost neodpovídá přesně přáním zákazníka — způsobeno nejednoznačností v chápání zadání).

7.4 Testování systému

- Systém je tvořen SW + HW + ostatními součástmi(lidé atd.).
- Testujeme celek.
- Typicky probíhá formou zkušebního provozu, který bývá časově náročnější než by bylo testování v umělých podmínkách.
- Chyby již se zpravidla neopravují, až v rámci údržby.

7.5 Návrh testovacích dat

Rozlišujeme tři způsoby nakládání s daty v případě, že se objeví chyba:

Hrubé násilí — trasování, dumpy paměti, obsahy proměnných a registrů. Všechny tyto informace necháme počítač vygenerovat, pak teprve začneme s hledáním příčiny.

Eliminace příčiny chyby — stanovíme hypotézu o příčině chyby.

Zpětný postup — protože se místo, kde se chyba zpravidla projevila, liší od místa vzniku chyby, jdeme v programu zpět.

7.6 Pomocné prostředky

7.6.1 Statická analýza

Statická analýza vypovídá o programu z jeho syntaxe, nevěnuje se způsobu práce programu za běhu. Touto metodou je možné zjišťovat například nepoužité proměnné nebo opakované přiřazování hodnot do proměnné, tedy možné zdroje chyb. Je to výhodný prostředek také ke kontrole vzhledem k různým standardům (přenositelnost jazyka).

7.6.2 Mutační testování

Mutační testování se používá k testování testovacích dat, ne programů. Je možné touto metodou určovat kvalitu testovacích dat, což je jinak dost těžko proveditelné. Myšlenka je zjednodušeně takováto: záměrně se udělá v programu chyba, když ji pak testovací data najdou, můžeme o nich prohlásit, že jsou kvalitní. Tato metoda se příliš nepoužívá, ale stojí rozhodně za pozornost.

7.6.3 Prostředky ke generování testovacích dat

Jde o nejrůznější CASE nástroje, které může použít například mutační testování atd.

8 Údržba

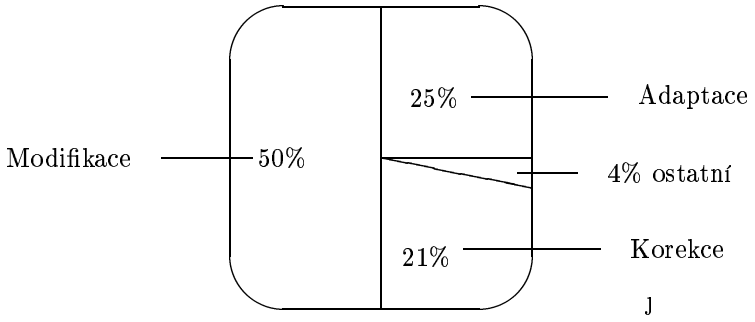
Při provozu se objevují nejrůznější požadavky na zásah do systému:

Korekce — v systému se objeví chyby, které je nutno odstranit.

Adaptace — přizpůsobení se změnám prostředí (změna HW).

Modifikace — změna požadavků na systém.

Prevence — zásah z důvodu správného a plynulého fungování systému, pro usnadnění budoucí údržby (nejvíce u OS).

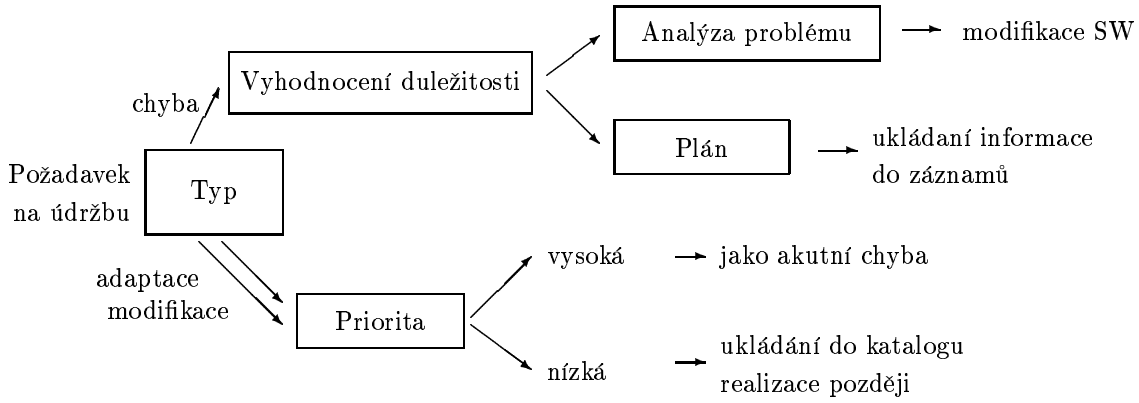


Obrázek 7: Vzájemný poměr druhů požadavků na zásahy do systému po uvedení do provozu

Rozlišujeme dva typy udržovaných systémů: starší a novější. Ty starší jsou většinou bez dokumentace, nestrukturované, máme pouze kód. Tento kód musíme nejprve vyhodnotit, pak teprve provádíme opravu — údržba je tímto nákladnější. Můžeme jen doufat, že existuje člověk, který takovému systému rozumí. Ty novější už bereme jako nový systém, vytvořený podle všech známých zásad.

Produktivita při údržbě je asi 40× nižší než při vývoji systému. Hlavní problémy souvisejí s počátečními fázemi vývoje systému.

Postup při údržbě je zobrazen na obrázku 8.



Neaktutní chyby, adaptace a modifikace s nízkou prioritou provedeme později,
např. při akutním požadavku

Obrázek 8: Rozhodování při požadavku na údržbu

Nad čím je třeba se zamyslet při údržbě:

- co se má změnit, udělat v systému jinak, aby odstranění chyby bylo jednodušší, nebo aby vůbec nevznikla — najít slabá místa.
- jaké jsou použité, nepoužité a potřebné prostředky v rámci údržby.
- co stálo nejvíce úsilí.
- co lze udělat pro prevenci.

ZÁZNAM O ÚDRŽBĚ obsahuje:

1. Identifikaci programu

2. Velikost programu
3. Použitý programovací jazyk
4. Datum instalace programu
5. Počet běhů
6. Počet selhání
7. Identifikace změny
8. Počet přidaných řádků
9. Počet odstraněných řádků
10. Úsilí věnované opravě
11. Datum změny
12. Autor změny
13. Počátek a konec údržby
14. Přínos změny

Údržba má vždy vedlejší efekt na:

- kód
- data (modifikace datových struktur)
- dokumentaci (přestává být aktuální ke změněnému stavu)

Preventivní údržba znamená:

- zásah do systému z důvodu snadnější údržby v budoucnu.
- při větších modifikacích úvahy zda modifikovat, znovu vytvořit či koupit nový systém.
  - ⋆ rozhodovací hlediska: ekonomická (cena, doba vývoje), zda je uživatel schopen precizněji specifikovat požadavky, máme-li zkušenosti s vývojem, budoucnost systému.

J