# 7. NÁVRH PROGRAMOVÉHO SYSTÉMU - STRUCTURED DESIGN

**Structured design** (SD) je **technika** pro návrh programových systémů, která navazuje na strukturovanou analýzu systému (uživatelský implementační model). Definuje vlastnosti **dobrého** programového systému a poskytuje návod, jak z uživatelsky orientované funkční specifikace (vyjádřené pomocí DFD) odvodit modulární strukturu programových systémů.

**Cílem** SD je vytvořit programový systém optimální kvality. Programový systém by měl být:

- **flexibilní a udržovatelný** (měl by předpokládat možnost budoucích změn. Měl by být srozumitelný pro každého, kdo chce provést nějakou změnu. Potom je možné odhadnout "cenu" dané změny, dopady změny na zbytek systému, dále je možné změnu provést a otestovat a doplnit do dokumentace),
- **spolehlivý** (měl by být úplně navržen a otestován. Nespoléhat na to, že systém bude spolehlivě fungovat, když se "zdá", že funguje),
- **uspokojivý a produktivní** (celý programový systém musí být navrhován "inženýrsky" s dokonalou disciplínou, stejně jako tvorba ostatních produktů - nejdříve se dobře navrhnou a teprve potom se realizují.Jinak se při tvorbě programových systémů se nejprve polovina času stráví děláním chyb a zbývající polovina jejich odstraňováním),
- **řiditelný** (programový systém musí mít srozumitelnou a pohodlnou dokumentaci. Musí navazovat na kvalitní analýzu "Testování" se nemá provádět až na konci vývoje systému, ale průběžně během celého vývoje).

**Výsledkem SD je tedy návrh automatizované části informačního systému** (tedy té části systému, která bude zpracovávána na počítači, programových systémů).

## 7.1. Structure chart

Nástrojem pro zobrazení struktury programového systému je **diagram** tzv. **"structure chart"** . Zůstaneme u anglického označení, protože překlad (strukturní diagram) by mohl být zaměňován se strukturním diagramem např. Jacksonovým (ten popisuje logiku jednoho programu, modulu). **Structure chart** (SCH) vyjadřuje **modulární strukturu programového systému** (ne strukturu jednoho modulu!). SCH zobrazuje :

- moduly,
- vztahy mezi moduly (jejich vzájemné volání a rozhra ní).

Nezobrazuje vnitřní logiku modulů ani počet vzájemného volání!

**Modulem** se rozumí - určitý počet příkazů, které zajišťují provedení nějaké činnosti. Modulem modul může být například procedura v Pascalu, funkce v C nebo Pascalu, program v COBOLu, podprogram ve Fortranu, procedura v ADA. Je to "volatelná" část programového systému, která má tyto atributy:

- **vstup** (údaje, které dostává od volajícího modulu) a výstup (údaje, které vrací volajícímu),
- **funkce** (co modul dělá, aby provedl transformaci vstupu na výstup),
- **způsob práce** (mechanika, vnitřní logika modulu - kód, pomocí kterého se provádí funkce - jak je prováděna transformace vstupu na výstup),
- **interní data** (vlastní pracovní oblasti modulu, na která se neodvolává žádná jiná část systému)

a další atributy, jako je název, volané moduly.

Vstupy, výstupy a funkce představují vnější pohled na modul, mechanika a lokální data představují vnitřní pohled na modul.

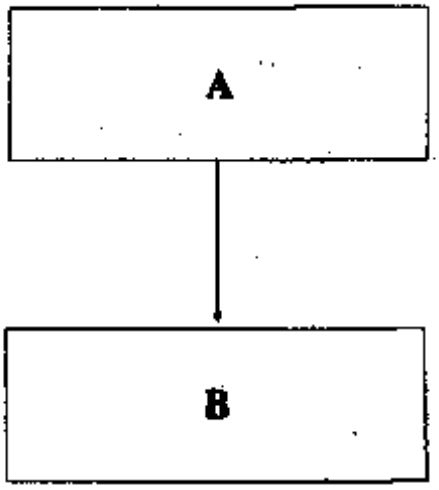
Structured design se zabývá vnějším pohledem na modul a na programový systém - tedy tím, co programový systém dělá, mikoli tím, jak to dělá (to je věcí strukturovaného programování).

Modul se graficky vyjadřuje obdélníkem, který má název, vyjadřující funkci modulu:

Předdefinovaný modul je modul, který již existuje a v programovém systému se používá (nebude se programovat, je ale nutné dodržet jeho rozhraní):


Spojení modulů

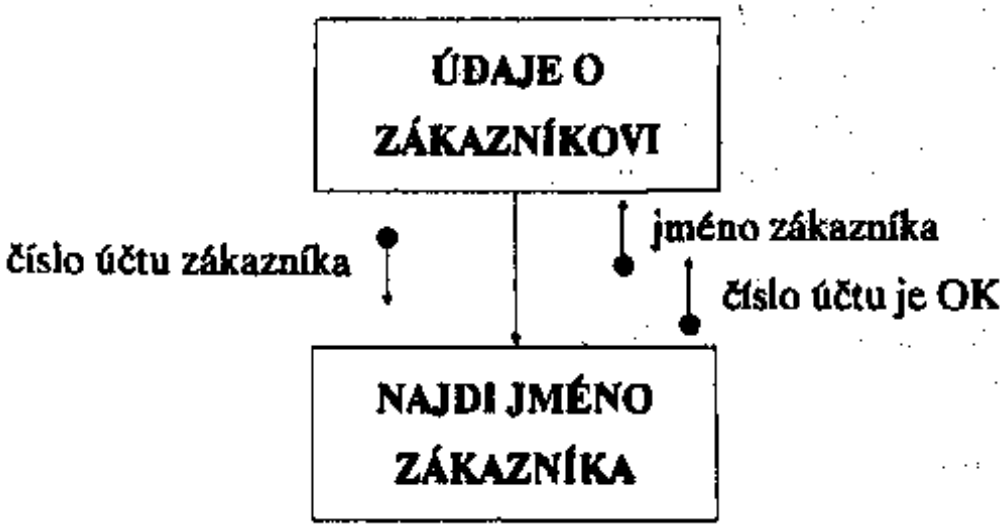
Moduly jsou uspořádány do struktury, která vyjadřuje jejich vzájemnou spolupráci a komunikaci - volání modulů.




modul A volá modul B
modul B vrací řízení modulu A
Neříká se kolikrát, ani jaká je vnitřní struktura A, ani B

Komunikace mezi moduly

Vyjadřuje předávaná data a signály mezi moduly (parametry volání)




Moduly si mezi sebou předávají data ( ●— ) a flagy (řídící nebo popisné signály - vyjadřují se plným kolečkem). Řídící flag říká modulu, co má

dělat, popisný říká, že nastala nějaká situace (například konec souboru apod.).

Structure chart popisuje rozhraní modulů, neukazuje počet ani pořadí volání modulů. Názvy parametrů jsou názvy, které používá volající modul. Název modulu musí vyjadřovat funkci daného modulu a všech jeho podřízených. Navíc je potřeba SCH doplnit o popis jednotlivých modulů (například pseudokódem) a popisem všech dat ve slovníku dat.

Structure chart spolu se specifikací modulů je zadáním pro programátory. Před programováním je ale potřeba ověřit kvalitu návrhu!

## 7.2. Postup structured designu

Výsledkem vývoje systému (strukturované analýzy a specifikace) je esenciální model systému. Popisuje, co musí systém dělat z hlediska uživatelů (jaké funkce musí provádět a jaká data k tomu potřebuje). Výsledkem uživatelského implementačního modelování a konfigurační analýzy je doplnění esenciálního modelu z hlediska požadavků na implementaci (rozdělení na ruční, dávkové, interaktivní a real-time části, výběr základního programového vybavení, technického vybavení a doplnění dalších omezení).

Funkční specifikace musí být řádně dokumentována:DFD a další části. Všechna data musí být popsána ve slovníku dat. Každá elementární funkce musí mít popis (minispecifikaci). Případně jsou k dispozici diagramy dialogu ("předávání řízení mezi obrazovkami").

Dalším krokem při vývoji procedurální části systému je navrhnout strukturu systému z pohledu realizace - takovou strukturu, která bude jasná a bude dobrým východiskem pro implementaci a následnou údržbu systému.

SD přímo navazuje na funkční specifikaci systému vytvořenou na základě analýzy událostí a reakcí na tyto události. Východiskem je popis systému pomocí DFD, a to úroveň jednotlivých "transakcí" (tj. těch funkcí v DFD, které vyjadřují reakce na události).

K odvození modulární struktury systému z funkční specifikace lze použít dvojici technik: **transakční** a **transformační** analýzu. Uvedeme nyní jednotlivé kroky návrhu modulární struktury

***Kroky návrhu modulární struktury***

*Poznámka: nejedná se o přesný jednoprůchodový postup, ale o zásady, je třeba se vracet a ověřovat.*

A. Transakční analýza.

B. Transformační analýza pro každou transakci.

C. Zpětné spojení jednotlivých oddělených částí , tak aby bylo možné systém implementovat jako celek.

**A. Transakční analýza**

vychází z vyhledání tzv. transakčního centra (funkce roz hodující na základě vstupních dat, která transakce se má provést - např. výběr z menu). SCH má na vrcholu modul "transakční centrum". Ten volá na základě vstupu pří slušné moduly, které provedou požadovanou transakci - ty to moduly odpovídají funkcím z DFD, které byly reakcí na události. Rozklad transakcí probíhá dále pomocí transfor mační analýzy.

**B. Transformační analýza**

je návod, jak převést jednotlivé transakce do SCH. Vý sledkem je vyvážený systém (systém, který má technologicky závislé moduly, které zpracovávají nezformátovaná a neupravená data, na spodní úrovni hierarchie, a moduly, které realizují vlastní transakci a pracují s "čistými" daty, na horních úrovních hierarchie). Postup vychází z identifikace "centrální transformace" v DFD, který je rozkladem transakce - proto název "transformační analýza".

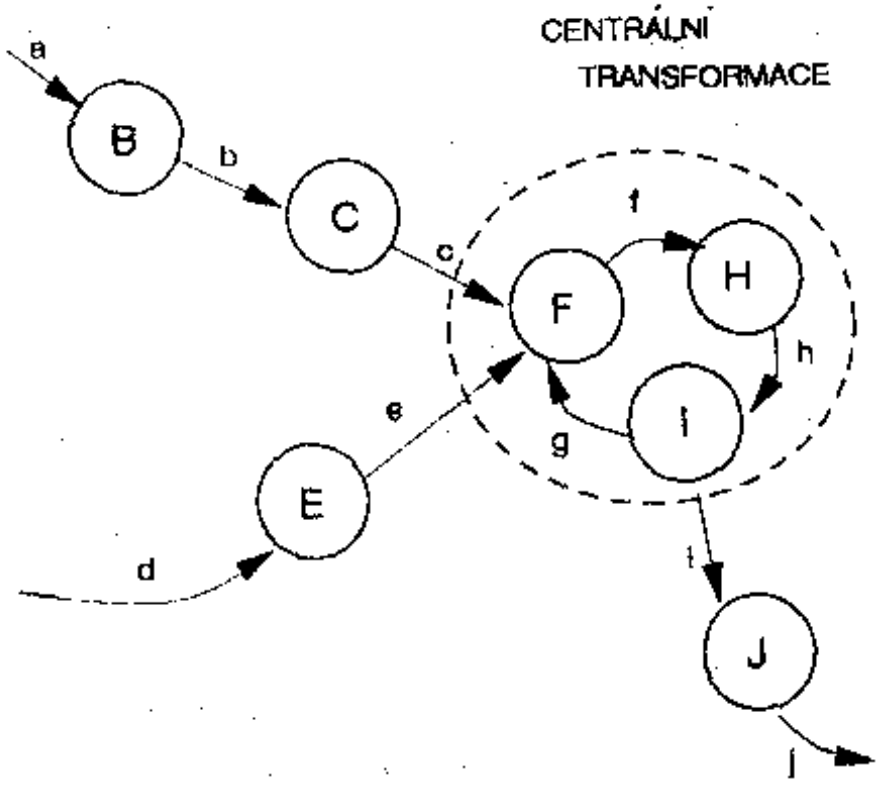
***Kroky transformační analýzy***

1) Pro každou transakci vytvořit DFD nižší úrovně (pokud již existuje - a to by měl - ve specifikaci, použije se. Tento krok nepopisujeme, spadá spíš do fáze funkční specifikace).
2) Nalezení centrální transformace v tomto DFD.
3) Převod DFD do hrubého SCH.
4) Ověření a úprava tohoto SCH podle pravidel a návodů pro "dobrý" design (spřaženost a soudržnost modulů, velikost modulů, odstranění redundancí, nalezení společných modulů, prověření dělení rozhodování ...).
5) Ověření, zda SCH splňuje všechny požadavky původního DFD.

Podrobnější vysvětlení jednotlivých kroků

ad 1) V DFD specifikaci nalezneme rozklad transakce, kterou chceme převádět do SCH.

ad 2) Nalezení centrální transformace v tomto DFD.




Obr. 22. SD - nalezení centrální transformace

Centrální transformace je jedna funkce nebo skupina funkcí (bublin DFD), které představují podstatu transformace nadřízené ("kdyby byl svět ideální"). Je nezávislá na konkrétní implementaci vstupů a výstupů. Všechny funkce, jejichž výstupní datové toky jsou vstupy do centrální transformace, připravují data do tvaru, v němž je centrální transformace potřebuje ("afferent" toky). Všechny funkce, které zpracovávají výstupní toky centrální transformace připravují data pro výstup mimo modelovanou transakci (funkci) (efferent toky).

Všechny funkce, tvořící centrální transformaci, se označí.

Při hledání centrální transformace jde vždy do jisté míry o subjektivní určení, zda jsou data již "v čistém tvaru", nebo nejsou. Ale centrální transformace slouží pouze jako východisko pro odvození hrubého SCH. Po uplatnění kriterií designu vznikne obdobný návrh, ať byla, nebo nebyla funkce, u které jsme pochybovali, zahrnuta do centrální transformace.

ad 3) **Převod DFD do hrubého SCH**

Úkolem transformační analýzy je převést DFD jedné transakce do SCH této transakce. Hlavní rozdíl mezi DFD a SCH je v tom, že hierarchie úrovní DFD vyjadřuje hierarchii podrobnosti, zatímco

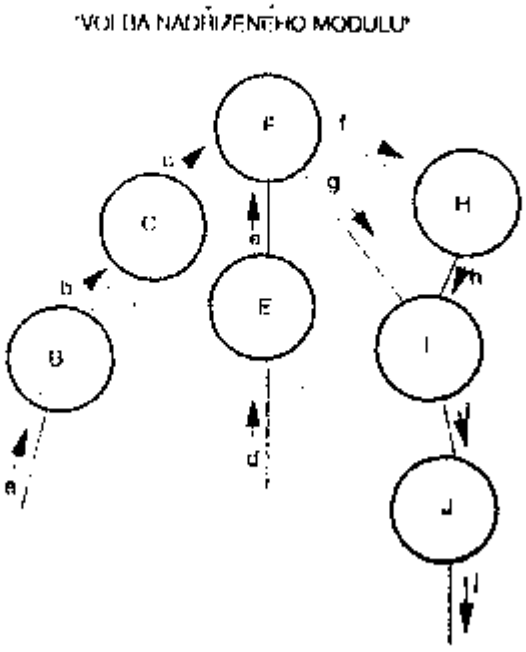
hierarchie SCH vyjadřuje strukturu řízení spolu se zpodrobněním. V DFD je nadřízená funkce složena z podřízených funkcí. Každá z nich odpovídá pouze za svoji činnost, zpracovává vstupní data a produkuje výstupní data nezávisle na ostatních funkcích. Ve SCH je vyjádřeno vzájemné řízení modulů - který modul volá který, co mu předává a co od něj přebírá, a zároveň s pomocí jakých modulů vykonává svoji funkci).

Aplikace transformační analýzy na DFD znamená zavedení řízení do zatím neřízeného problému. Stejně jako v životě se řídící pracovník buď volí v rámci daného kolektivu, nebo se dosadí vedoucí nový, je tomu tak i při transformační analýze. Buď se v nalezené centrální transformaci vyhledá funkce, která je vhodným kandidátem na "vedoucího" (tj. kořen SCH), nebo se dosadí nový "vedoucí" (tj. kořenem se stane nový modul).

"Volba kořene SCH" v rámci centrální transformace probíhá takto:

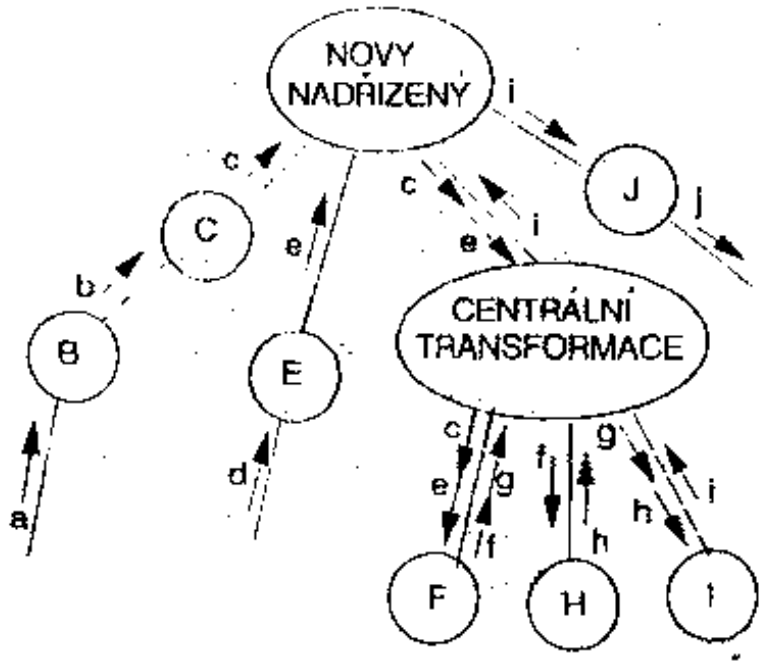
- hledá se funkce, která má hodně vstupů a výstupů, a jinak je dost jednoduchá (vypadá, jako by koordinovala činnost ostatních bublin), nebo
- se použije "geometrický" přístup - vybere se funkce, která je přibližně ve středu centrální transformace.

V případě, že existuje několik funkcí s těmito vlastnostmi, je dobré zkusit všechny kandidáty na "kořen SCH" a vzít nejlepší výchozí návrh.




Obr. 23. SD - volba kořenového modulu z již existujících

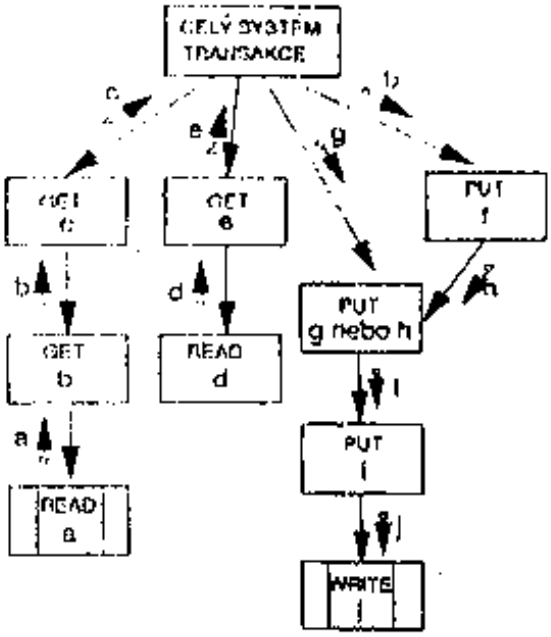
"Dosazení kořene SCH" se provede v případě, že se nenajde vhodný kandidát na "kořen" mezi funkcemi centrální trans formace. Všechny afferentní (vstupní) větve se poskládají vlevo pod kořen, centrální transformace do středu a všechny efferentní (výstupní) větve vpravo.




Obr. 24. SD - volba nového kořenového modulu

*Příklad volby kořene:*

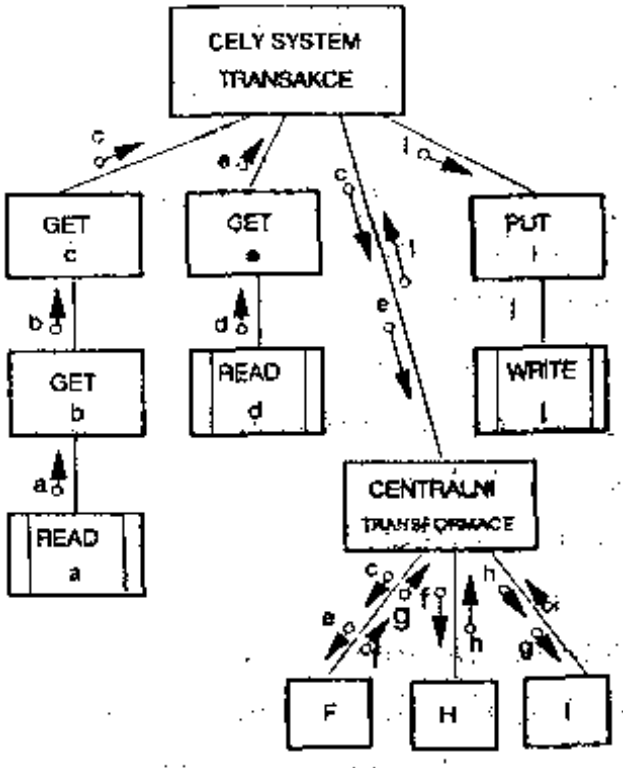
*Funkce v DFD si můžeme představit jako korálky navlečené na niti. Vezmeme funkci, kterou jsme zvolili za kořen a korálky zvedneme. Ostatní funkce se poskládají pod "vedoucího" - vznikne hierarchická struktura - překreslí se do SCH.*




Obr. 25. SD - hrubý SCH s volbou kořene z původních transformací

Tuto přeměnu je potřeba dodělat. V DFD není vidět řízení. Šipky v DFD ukazují směr toku dat a nemusí nutně korespondovat s řízením - tedy se šipkami mezi moduly v SCH. Je potřeba vytvořit názvy modulů. Opět nemusí být shodné s názvy funkcí v DFD. Název funkce vyjadřuje pouze, co dělá daná funkce. Název modulu vyjadřuje, co dělá daný modul a všechny moduly mu podřízené. Přidáním modulů pro čtení a zápis z okolí systému a pro přístup k datům vznikne první hrubý SCH.

Jak poznat, že zvolený "kořen SCH" je ten pravý? Vždy je dobré zkusit několik variant a vybrat strukturu, která vyhovuje nejvíc. Častou vlastností špatného výchozího návrhu je tzv. inverze moci, podřízený modul posílá nadřízenému informace, co má dělat. Pak je dobré posunout na místo kořene ten modul, který takové řídící informace vysílá (viz další krok).




Obr. 26. SD - hrubý structure chart s dosazením nového kořene

ad 4) **Ověření a úprava SCH podle pravidel dobrého designu**

Hrubý SCH je nutné upravit tak, aby odpovídal pravidlům structured designu. Tak vznikne návrh, který bude splňovat všechny principy "dobrého" návrhu. Je potřeba provést rozdělení modulů dle pravidel pro vydělování funkcí (factoring), ověřit a upravit moduly z hlediska soudržnosti a spřaženosti, odstranit stavovou paměť způso-

benou výměnou nadřízenosti a podřízenosti modulů atd. Přehled těchto principů je v následující kapitole.

Úprava hrubého SCH zahrnuje:

- přidání modulů čtení a zápisu pro přístup k okolí sy stému a k souborům (pokud je to nutné, přidejte klíče pro čtení jako parametry volání těchto modulů);
- rozdělení a reorganizaci vstupní a výstupní větve SCH. Při úpravách je třeba dodržet tvar (vyváženost) systému. Je dobré nemít moduly, které pracují s více než se dvěma různými datovými strukturami;
- rozdělení centrální transformace, jestliže je příliš složitá (lze použít nižší úrovně DFD);
- přidání modulů na obsluhu chyb;
- přidání inicializačních a závěrečných činností (pokud je to potřeba);
- ověření názvů modulů (zda každý vyjadřuje roli modulu v rámci hierarchie);
- doplnění všech popisných (případně i řídících) signálů (flagů) - ve SCH jsou nutné, v DFD se nevyjadřují - například "konec vstupu", "chyba numeričnosti" apod.;
- ověřit všechna kriteria návrhu.

ad 5) **Ověření, zda SCH splňuje všechny požadavky původního DFD.**

Tento krok je rozhodující. Důvodem pro provedení transformační analýzy není udělat transformační analýzu, ale rychleji odvodit SCH, který:

- správně implementuje specifikované funkce
- a zároveň splňuje kriteria udržovatelného systému.

Při provádění hrubé transformační analýzy je dobré si představit kód (nebo pseudokód) každého modulu. Pokud bu de u většiny modulů tato představa hrozná, je lepší zku sit jiný kořen SCH a důsledně dodržovat kriteria structu red designu.

Výsledek je důležitější, než prostředky, kterými se k němu dojde. S trochou praxe lze návrh SCH udělat rovnou. Ale ať dojdeme k návrhu jakkoliv, je vždy nutné ověřit, zda správně implementujeme požadavky vyjádřené v DFD a minispecifikacích funkcí.

Pro každý modul je také potřeba doplnit jeho specifikaci a doplnit popis všech nových dat do slovníku dat.

**C. Zpětné spojení jednotlivých oddělených transakcí**

Provedli jsme transakční analýzu a získali transakční centrum. Pro každou transakci jsme odvodili hrubý SCH, který jsme pak zjemňovali a měnili až do uspokojivé podoby. Nyní zbývá provést poslední krok návrhu modulární struktury - zpětné spojení jednotlivých oddělených transakcí. Tento krok je nutný, aby vznikl úplný systém. Jedná se v podstatě o návrat k dalším činnostem "transakční" analýzy - dotvoří se struktura volání transakcí (lze použít strukturu dialogu vytvořenou v rámci uživatelského implementačního modelování). Odvozené SCH transakcí se poskládají pod moduly typu "transakční centrum". Navíc je dobré prověřit celý systém a nalézt moduly společné několika transakcím.

Při návrhu modulární struktury již nechceme a ani nemůžeme odfiltrovat vlastnosti implementačního prostředí. Např. u víceuživatelského systému je třeba respektovat vlastnosti příslušného transakčního monitoru a navíc kódovat jednotlivé transakce jako reentrantní. To však je věcí až implementace systému.

## 7.3. Shrnutí odvození Structure Chart

K vytvoření množiny SCH ze strukturované specifikace je třeba uskutečnit následující kroky:

1) Rozdělit množinu DFD celého systému podle typů transakcí, tak, aby pro každou transakci vznikla množina DFD (pokud bylo provedeno rozčlenění systému dle událostí během strukturované analýzy, je možné pokračovat přímo).
2) Převést DFD každé transakce do jednoho SCh pomocí prostředků transformační analýzy
   - 2.1 Identifikace centrální transformace v DFD dané transak ce oddělením "afferent" větví DFD (obsahují a upravují vstupy) a "efferent" větví (obsahují a upravují výstupy)
   - 2.2 Vytvoření prvního hrubého SCh a to:
     - 2.1.1 BUĎ - zvolením kořene z transformací (bublin) v rámci centrální transformace
     - 2.1.2 NEBO - dosazením nového kořene nad centrální transformaci a aferentní a eferentní toky.

2.3 Využít procesy uvnitř centrální transformace a procesy uvnitř nižších úrovní DFD jako vodítko k odhalení (vydělení) nižších úrovní modulů.

2.4 Zpodrobnit a upravit SCh podle principů strukturovaného designu.

2.4.1 Přidat moduly čtení, zápisu, přístupu do databáze...

2.4.2 Vydělit (factoring) a reorganizovat moduly (ale dát si pozor na vyváženost SCh - aby nebyl řízen ani fyzickými vstupy, ani fyzickými výstupy)

2.4.3 Přidat moduly ošetření chyb

2.4.4 Přidat inicializační a závěrečné činnosti

2.4.5 Ověřit, zda názvy modulů korespondují s jejich rolí v celé hierarchii

2.4.6 Přidat nezbytné indikátory (= flagy)

2.4.7 Ověřit a případně provést další vydělení modulů (factoring), prověřit soudržnost, state memory, korespondenci datových struktur a ostatní kriteria designu (spřaženost, obecné a speciální moduly apod.)

2.5. Ověřit, že návrh bude skutečně fungovat - tj. že bude odpovídat specifikaci (ověření s DFD).

3. Spojit dohromady všechny SCh všech typů transakcí pod transakční monitor nebo pod "transakční centrum", aby každý SCh mohl být spuštěn, kdykoliv přijde stimul příslušného typu (nebo kdykoli je vybrán z menu).

## 7.4. Kriteria structured designu

Ve SD se snažíme **zlepšit kvalitu návrhu** z hlediska

- spřaženosti mezi moduly (snažíme se o co nejslabší stupeň spřaženosti),
- soudržnosti uvnitř jednoho modulu (snažíme se dosáhnout co největší soudržnosti činností uvnitř modulu) a
- ostatních kriterií kvality dobrého návrhu:
  - vydělování funkcí z "velkých" modulů (factoring),
  - tvar systému,
  - dělení rozhodování,
  - kontroly a úpravy dat,

- ošetření chyb a chybové zprávy,
- počet přímých podřízených modulu (fan-out),
- počet přímých nadřízených modulu (fan-in),
- korespondence datových struktur,
- pravidla inicializace a závěrečných činností,
- odstranění redundancí,
- odstranění stavové paměti,
- tvar systému,
- filtrující a datové moduly.

*Spřaženost*

Spřaženost je míra možnosti, že se chyba v jednom modulu projeví jako chyba v jiném modulu, nebo že se změna v jednom modulu projeví jako chyba v jiném modulu nebo vyvolá nutnost změny v jiném modulu.

Principy spřaženosti (jak jsou moduly propopojeny?):

- vytvořit **úzké spojení** (na rozdíl od širokého),
  (je lepší, když si moduly předávají 2 datové prvky namísto 15ti (široké spojení));
- vytvořit **přímé spojení** (na rozdíl od nepřímého), (interface mezi moduly je pochopitelnější, jestliže mu člověk rozumí hned, aniž by musel hledat další informace);
- vytvořit **lokální spojení** (na rozdíl od vzdáleného), (informace potřebná k pochopení vztahu mezi dvěma moduly je přímo součástí rozhraní. Příkladem vzdálené vazby je vazba dvou modulů přes globální data - informace o vazbě může být stovky řádků kódu vzdálená ve volaném nebo volajícím modulu. Nebo jsou počáteční hodnoty dat nastavovány dlouho před tím, než jsou skutečně používány. Tyto "vzdálenosti" zvyšují nepochopitelnost rozhraní);
- vytvořit **zřejmé spojení** (na rozdíl od nezřetelného), (př. nezřetelného zamlžujícího rozhraní - modul A v assembleru modifikuje obsah tabulky v modulu B. Jestliže někdo chce zjistit, jak se v této tabulce dané hodnoty vyskytly, musí se hodně snažit);
- vytvořit **flexibilní spojení** (na rozdíl od pevného).

*Typy spřaženosti*

- "Normální"
  - datová,

- strukturou záznamu (stamp),
- řízením.

- Globální
- Obsahová (patologická)

*Soudržnost*

Soudržnost je míra těsnosti funkčních vztahů mezi prvky v rámci jednoho modulu (prvky se rozumí příkazy, skupiny příkazů, deklarace dat nebo volání jiného modulu - jakákoliv část modulu, která provádí nějakou činnost nebo definuje data).

Návrh by se měl skládat z co nejvíce soudržných modulů. Soudržnost těsně souvisí se spřažeností. Čím jsou soudržnější moduly uvnitř, tím jsou méně spřažené mezi sebou, a naopak.

*Typy soudržnosti*

- Funkční soudržnost
- Sekvenční soudržnost
- Komunikační soudržnost
- Procedurální soudržnost
- Časová soudržnost
- Logická soudržnost
- Nahodilá soudržnost

*Ostatní kriteria dobrého návrhu.*

Factoring - vydělení funkce obsažené jako kód v jednom modu lu do jiného modulu z důvodu:

a) redukce velikosti modulu,
b) dodržení výhod klasického návrhu shora dolů a tak (systém se zjednoduší pro pochopení a změny),
c) odstranění existence stejné funkce z více modulů,
d) oddělení práce (výpočty a kontroly) od řízení (volání a rozhodování),
e) vytvoření obecně využitelných modulů,
f) zjednodušení implementace.

Je dobré dosáhnout vysokého stupně rozdělení.

Dělení rozhodování (decision split) - existence dělení rozhodnutí se pozná podle toho, že data, která slouží jako podklad pro rozhodnutí, a data

potřebná pro provedení rozhodnutí existují v různých modulech. Tato vlastnost se většinou projeví nežádoucími "trampujícími" indikátory. Rozdělení rozhodování je potřeba odstranit. Obě části, jak samotné rozhodnutí, tak činnosti, které z něho vyplývají, je dobré dát co nejblíže k sobě. Nejlépe do jednoho modulu.

Tvar systému - návrh vede k vytvoření "vyváženého systému" - tj. moduly na horních úrovních Structure Chartu pracují s daty v logickém tvaru (spíš než ve fyzickém tvaru). Na vstupní (afferent) straně jsou data zkontrolována a odblokována - "nahoře" nejsou tedy závislá na tom, jak fyzicky vstupují. Na výstupní (efferent) straně jsou data co nejméně závislá na určitém výstupním formátu. Fyzické charakteristiky vstupů a výstupů jsou na spodních úrovních větví "afferent" a "efferent".

Výpisy chyb - chyba má být hlášena modulem, který rozezná že jde o chybu. Je výhodnější mít chybové zprávy na jednom místě. Možná řešení modul pro výpis včetně chybových hlášení, nebo hlášení mít uložena v souboru, a vytvořit modul, který zná strukturu tohoto souboru) a předávat chybové kódy.

Kontroly a úpravy dat - toto kriterium říká, v jakém pořadí kontrolovat části vstupních dat:

- známé před neznámým,
- syntaxi před sémantikou,
- jednoduchá kontrola před křížovou,
- kontrola vnitřní před externí.

Stavová paměť - existence stavové paměti se projeví existencí interních dat modulu, která se nesmějí změnit od jednoho volání k druhému (tzv. statická či stavová data). Pokud je to možné, je dobré zabránit vytváření stavových dat (jinak není možné vícenásobné použití modulu - modul má nepředvídatelné chování).

Korespondence datových struktur - přizpůsobení programové struktury modulu datovým strukturám, které modul zpracovává - programování a údržba modulů je jednoduš ší, pokud programová struktura vyplývá ze struktur zpracovávaných dat.

Information cluster (datový modul, filtrující modul) - mno žina modulů, které mají výhradní právo přístupu k urči tým datům (data mají složitou strukturu nebo jsou cit livá na ochranu nebo jsou závislá na zařízení apod.).

Inicializační a závěrečné činnosti - inicializaci každé fun kce je dobré provést co nejpozději a závěrečné činnosti každé fce co nejdříve, neboli

inicializujte a prováděj te závěr každé funkce tak nízko v hierarchii, aby neby la nutná stavová paměť.

Omezené x všeobecné moduly

*omezený modul* - provádí málo potřebnou speciální funkci,
- pracuje s omezenými datovými hodnotami, typy nebo strukturami,
- předpokládá, že bude použit v nějakém konkrétním kontextu.

Pokud se navrhuje modul s obdobnou funkcí, jaká už je navržená, je dobré již hotový modul zobecnit a použít pro realizaci dané funkce.

*všeobecný modul* (příliš všeobecný)
- provádí až příliš rozsáhlou činnost (která není třeba),
- pracuje s mnoha typy dat, hodnotami ne bo složitými strukturami,
- čte nebo přebírá parametry, které se velmi málo pravděpodobně budou někdy mě nit (konstanty).

Redundance - je potřeba odstranit redundanci funkcí, hodnot dat i datových struktur.

Fan-out - počet přímých podřízených modulu. Použít pravidla "magického" $7 \pm 2$.

Fan-in - počet přímých nadřízených modulů daného modulu. Je dobré mít vícenásobně použitelné moduly, tedy moduly s velkým počtem volajících. Velké "fan-in" se zajistí pečlivým factorin gem a odstraněním redundancí. Pozor na soudržnost a rozhraní (stejný počet parametrů a jejich typů ve všech rozhraních).

***Implementace takto navrženého programového systému (SCH).***

Činnosti, které je potřeba udělat:

- sdružení modulů do programů,
- sdružení modulů do běhů (load units),
- implementace
  - top-down versus bottom-up,
  - inkrementální versus tradiční.

## 7.5. Závěr ke kapitole Structured Design

Mezi esenciální (resp. uživatelskou implementační) specifikací systému a jeho implementací je potřeba udělat ještě hodně práce. Část jsme právě

popsali. Před vytvořením struktury programového systému pomocí SCH je navíc nutné konfigurovat systém na jednotlivé procesory, rozhodnout o umístění sdílených dat mezi nimi. Dále je nutné rozhodnout o rozdělení jednotlivých funkcí a dat do různých úloh (tasks). A pro každou úlohu vytvořit hierarchickou strukturu modulů pomocí SCH.

Z vlastností zvolené implementační technologie často vznikne potřeba přidat do implementačního modelu další procesy (funkce) a uložená data. Například jsou potřeba procesy pro kontrolu chyb, opravy a ověřovací činnosti, které nejsou v esenciálním modelu. Nebo se přidají další procesy pro přenos datových toků mezi procesory apod. Teprve potom je možné začít programovat.