Softwarové inženýrství
(státnicová otázka 2–8)

Ladislav Dobiáš

10.2.2000

Obsah

1 Zadání

Základní pojmy softwarového inženýrství (SI), specifikace požadavků na software, typy specifikací. Strukturovaná a objektově orientovaná analýza systému a používané metodiky.

(Předměty: SI, CIM, CSD, KTS, PPV, MEP, PP)

2 Základní pojmy

Základním zdrojem jsou skripta a přednášky [3, 2].

Termín *softwarové inženýrství* (SI) je používán v souvislosti s řešením velkých programových systémů, na rozdíl od „programových technik“, používaných pro řešení menších problémů.

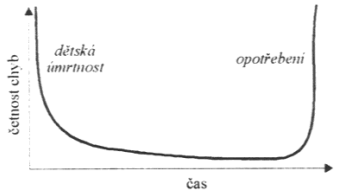
Asi nejdůležitějším pojmem SI je „životní cyklus“, který je vysvětlen v kap. 2.4.

2.1 Charakteristiky software

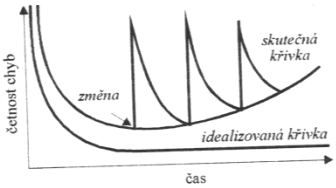
„Čím se software liší od jiných lidských produktů?“ – Software je více logický, než fyzický.

1. Software je vyvíjen a řešen inženýrskými pracemi, není vyráběn v klasickém slova smyslu. Předpokladem vysoké kvality je v obou případech dobrý návrh, při výrobě však nalezneme další faktory kvality, které při produkci software nemají význam. Cena software se koncentruje do inženýrských prací. Nelze je tedy řídit způsoby, které se osvědčily v klasických výrobních procesech.

2. Software se fyzicky „neopotřebuje“. Na obr. 1 je znázorněna křivka četnosti chyb v čase pro hardware (HW) a na obr. 2 pro software (SW). HW má na počátku a na konci svého života vysokou chybovost. Naproti tomu SW není poškozován vlivem prostředí — to ukazuje idealizovaná křivka.



Obrázek 1: Četnost chyb HW



Obrázek 2: Četnost chyb SW

Avšak SW se stále obtížněji přizpůsobuje, a díky opravám má skutečná křivka jiný tvar. HW lze snadno opravit výměnným způsobem. SW ale nemá náhradní díly, vyžaduje jiný způsob údržby.

3. Většina software je vyrobena „na míru“ podle přání zákazníka, málo softwaru je sestaveno z předem existujících komponent.

2.2 Programování ve velkém

Předmětem SI je vývoj softwaru na zakázku. Jde většinou o řešení rozsáhlých programových celků, které řídí, ovlivňují či spoluvytvářejí složitý systém.

Objasníme si dva termíny:

programování v malém se zabývá řešením jednotlivých modulů, uplatní se při něm programovací techniky pro návrh dat a algoritmů

programování ve velkém se zabývá dekompozicí projektu na menší celky, až se posléze dospěje k zadání modulů, řešitelných technikami programování v malém

Při řešení velkého projektu se tedy uplnatní jak programování ve velkém, tak i programování v malém.

Zakázka je obvykle řešena v krocích (etapách), které jsou přibližně znázorněny na obr. 3.

2.3 Řízení prací při pogramování ve velkém

Řízením projektu se zabývá manažer, jehož hlavní směry činnosti jsou alespoň pro představu naznačeny v bodech:

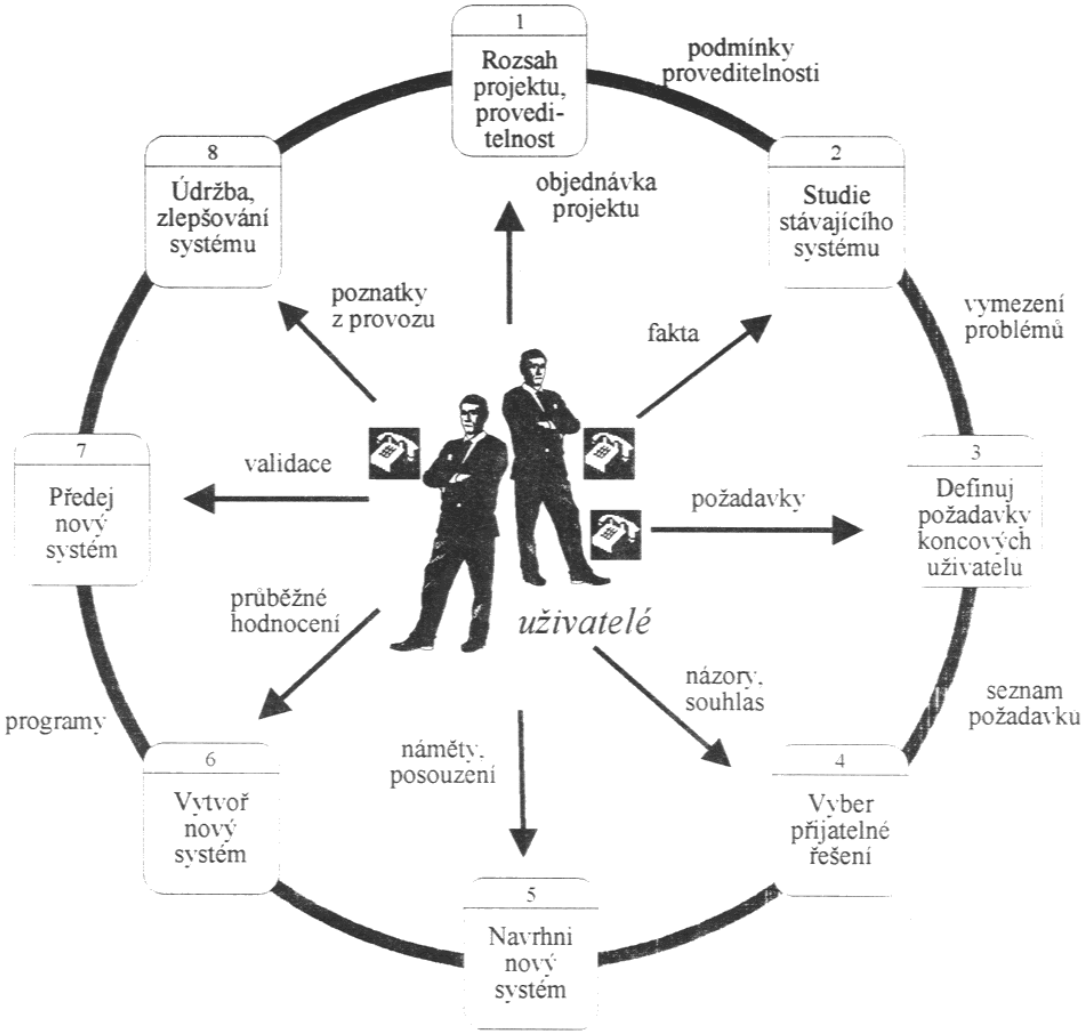
1. komunikace se zadavateli
2. plánování projektu
3. řízení rozpočtu
4. výběr řešitelů
5. kontrola stavu projektu
6. prezentace výsledků

2.4 Životní cyklus

Životní cyklus začíná prvotní představou o programu a končí, až se program přestane používat.

Životní cyklus programu obsahuje následující etapy:

1. **Specifikace problému:** zadání problému, specifikace problému, návrh komunikace s programem, testování specifikace.
2. **Analýza:** vytvoření logického modelu řešeného systému, záznam logického modelu pomocí grafických technik.
3. **Návrh (programování ve velkém):** rozklad na podproblémy, návrh metody řešení, návrh podpůrných prostředků.
4. **Implementace (programování v malém):** návrh reprezentace lokálních dat, návrh algoritmů, zápis řešení v programovacím jazyce, ladění programu, syntéza komponent.



Obrázek 3: Řešení systémů na zakázku

5. **Dokumentace produktu:** dokumentace programu, příručka pro užívání, dokumentace pro údržbu a modifikace.
6. **Testování produktu:** validace či verifikace produktu proti specifikaci a dokumentaci produktu.
7. **Provoz a údržba produktu:** oprava chyb u produktu a dokumentace, vývoj a údržba verzí atd.

Ze zkušeností vyplývá, že vývoj programu se neobejde bez zpětných kroků. Zhlediska efektivity je účelné vyloučit zpětné kroky přes více etap. I když zpětná smyčka přes celý cyklus má někdy smysl — např. pro ovládání zcela nových prostředků a potřebujeme nejdříve získat zkušenost s provozem, abychom mohli zodpovědně navrhnout zadání.

Z tohoto lze odvodit několik modelů životního cyklu, na jejichž základě lze nalézt:

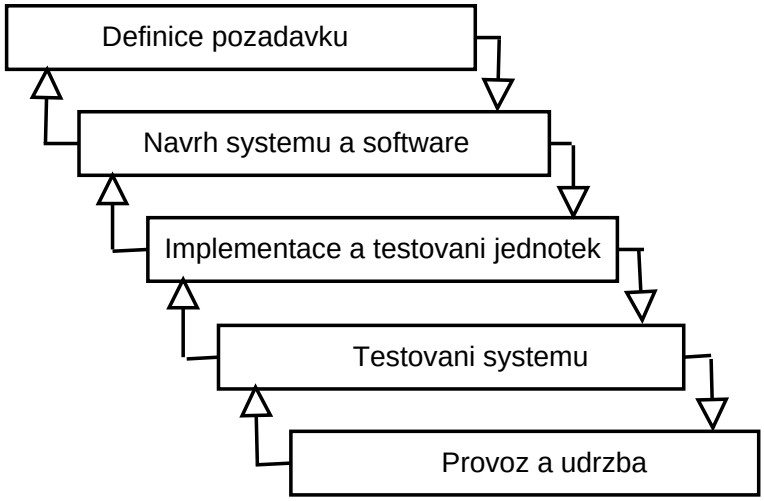
- nové technologie zvyšující produktivitu programátora

- odhadnout časové nároky na ukončení jednotlivých etap
- ohadnout celkovou ceny projektu

Základní modely životního cyklu si ukážeme v následujících kapitolách.

2.4.1 Model vodopád

Toto je základní model, který je složen z činností, které na sebe navazují a vzájemně se neprolínají. Tradiční model vodopád je na obr. 4.



Obrázek 4: Vodopád

Tento model má některé nevýhody, např.:

- reálné projekty se podle něj zřídka řídí
- pro uživatele/zadavatele je často velmi obtížné přesně specifikovat požadavky
- zákazník musí být trpělivý — provozuschopná verze bude k dispozici až po delší době, v posledních fázích řešení

Každý z těchto problémů je závažný. Přesto však má klasický životní cyklus pevné místo v SI, neboť slouží jako šablona pro řazení metod analýzy, návrhu, kódování, testování a údržby. I přes všechny zmíněné nedostatky je podstatně lepší, než náhodný, metodicky neřízený přístup k řešení problému.

2.4.2 Prototypování

Na rozdíl od modelu vodopád, který předpokládá, že výchozí požadavky na SW se nemění, nebo jen málo, tento model počítá s tím, že nemáme podrobně analyzovaný systém.

> Prototyp je částečnou implementací produktu (části produktu) v logické nebo fyzické formě, která prezentuje všechna vnější rozhraní.

Pomocí prototypování si obě strany — zákazníci a řešitelský tým — vyjasní nejen požadavky, ale i jejich interpretaci.

Prototypování bývá zahrnuto v různých jiných modelech. Obsahuje následující kroky:

1. sběr a analýza požadavků
2. rychlý návrh
3. tvorba prototypu
4. vyhodnocení prototypu zákazníkem
5. vylepšení návrhu prototypu
6. jesliže není zákazník spokojen s prototypem, pak opakuj od bodu 2
7. pokud je zákazník spokojen, pak proveď úplný návrh

Kritickým faktorem je rychlost obrátky při návrhu a tvorbě prototypu.

Prototypování lze většinou aplikovat přivývoji menších systémů a na úrovni subsystému. Kompletní prototypování rozsáhlého systému je obtížné a neekonomické.

2.4.3 Spirálový model

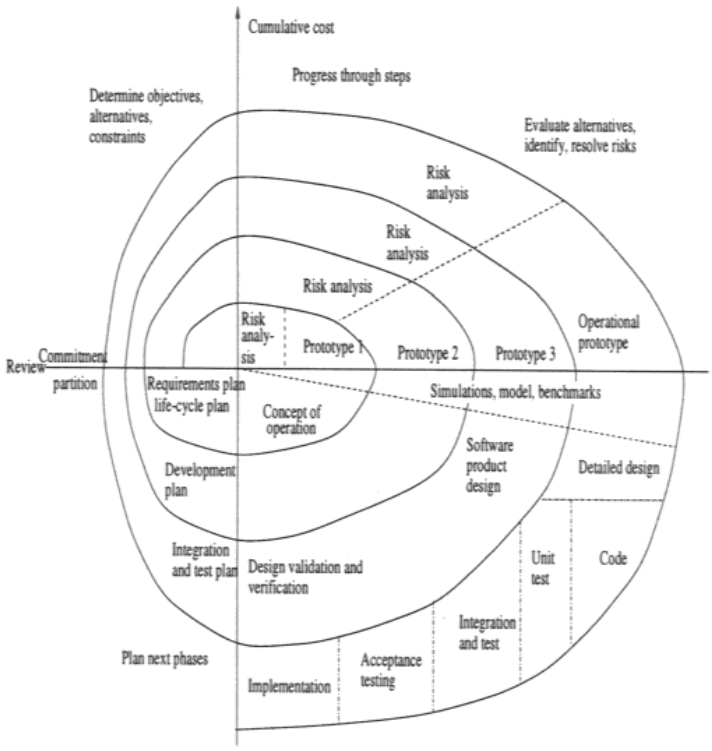
B. W. Boehm v 80. letech vytvořil spirálový model životního cyklu. Model je založen na kombinaci prototypování a analýzy rizik. Práce na projektu jsou organizovány tak, aby se cílevědomě minimalizovaly problémy spojené s klasickým modelem vodopád. Jednotlivé kroky při vývoji systému se ve spirále opakují, ale pokud možno na vyšším stupni zvládnutí problématiky — viz obr. 5.

Spirála začíná uprostřed, směrem ven postupuje čas (a peníze). Každá fáze vodopádu je zde řešena shodným postupem, který se skládá z dílčích kroků:

1. určení předmětru řešení, alternativ a omezení
2. vyhodnocení alternativ, identifikace a řešení rizik
3. vývoj produktu pro danou úroveň
4. plánování příští fáze

I tento model má nevýhody, např.

- nevyhovuje při řešení SW na zakázku — kde jde o termíny a cenu, což se zde obtížně zjišťuje
- závislost na rizikové expertýze, která tvoří páteř modelu



Obrázek 5: Spirálový model života SW

3 Specifikace požadavků

Specifikace požadavků obvykle probíhá v několika krocích. Výstupem každého kroku je dokument, který slouží jako podklad pro další řešení. V prvém kroku je vytvořen návrh požadavků, kterému se říká *neformální specifikace* nebo též *odborný článek*.

Přestože odborný článek tvoří prvý psaný dokument, použití přirozeného jazyka neposkytuje přesnou formu vyjádření. Proto se nedoporučuje, aby odborný článek tvořil jediný základ smlouvy mezi zadavatelem a řešitelem.

Na základě odborného článku je možno vytvořit funkční specifikaci programového systém. Tento dokument musí být mnohem přesnější než odborný článek, a proto je žádoucí použít pro jeho vyjádření mnohem formálnější notaci.

3.1 Typy specifikací

Specifikace mohou být funkční — tedy určující, jak bude program fungovat, a nefunkční. Jak jsme již naznačili, funkční specifikace jsou formální a neformální. Také jsou požadavky na uživatelský vzhled.

3.1.1 Neformální požadavky

Neformální specifikaci úlohy formou odborného článku si ukážeme na příkladu na formátování textu podle Petra Naura:

> Je dán text, který se skládá ze slov oddělených znaky SP (mezera) nebo NL (nový řádek). Úloha řádkového formátování spočívá v rozdělení textu do posloupnosti řádků, dle následujících pravidel:
>
> 1. řádky mohou být rozděleny pouze mezi slovy
> 2. každý řádek je co nejvíce zaplněn
> 3. žádný řádek neobsahuje více jak MAX znaků

Toto zadání se zdá na první pohled úplné a bezesporné, lze při bližším zkoumání zjistit nedostatky, např.:

- v zadání chybí explicitní vyjádření faktu, že výstupní text by měl obsahovat stejná slova jako vstupní a ve stejném pořadí
- není uveden případ, kdy slovo je delší než MAX — má to být chyba nebo se má rozdělit a kde?
- co dělat s více oddělovači? Ponechat či nahradit jediným
- není explicitně požadováno, aby ve výstupním textu byla slova oddělena oddělovači

Tímto jsme si ukázali, že stvořit neformální specifikaci není jednoduchý problém.

3.1.2 Formální požadavky

Nedostatky použití přirozeného jazyka při specifikaci lze odstranit zápisem ve formálním jazyce, jehož sémantika je jednoznačně definována.

Poměrně univerzálním jazykem je jazyk predikátové logiky, který je ovšem v praxi ne příliš použitelný.

Často lze využít i jiné prostředky známé z teorie automatu nebo předkladu.

Dalším příkladem formální specifikace může být tzv. Bacus-Naurova forma.

Využití formální specifikace je např. při validaci či verifikaci. Takové využití je ovšem možné pouze za předpokladu existence podpůrných nástrojů pro zpracování formálních specifikací.

3.1.3 Nefunční požadavky

Nefunkční požadavky lze rodělit takto:

- požadavky na řešení
  - podmínky dodání
  - implementační požadavky
  - použití standardů
- pžadavek na výrobek
  - přenositelnost
  - spolehlivost
  - použitelnost
  - efektivita
    - požadavky na výkon
    - požadavky na paměť
- vnější požadavky
  - legislativní požadavky
  - cenové omezení
  - schopnost spolupráce

4 Analýza systému

Nejdříve si řekneme pracovní definici pojmu „systém“.

> Systém je množina vzájemně souvisejících objektů či komponent, na kterou nahlížíme jako na celek a která byla vytvořena pro určitý účel; systém je ohraničený a to, co je vně této hranice, se nazývá okolí, s nímž systém interaguje prostřednictvím svých rozhraní.

Základní úlohou analýzy systému je nacházení faktů, zákonitostí a omezení v systému. Prvním úkolem při analýze je stanovení hranice systému. Softwarový produkt je pak správným či řídícím systémem nad systémem základním (pozor na směšování pojmů).

4.1 Metody návrhu systému

Dva možné přístupy

a) zdola nahoru (bottom-up) — založeno na generalizaci: začíná identifikací a analýzou několika jednoduchých struktur, snaží se najít jejich společné charakteristiky a chování a zobecnit je tak, aby původní byly jejich speciálními případy.

b) shora dolů (top-down) — založeno na specializaci; začíná na úrovni komplexního systému, který postupně dekomponuje na subsystémy, které jsou dále dekomponovány atd... Analytický proces je zobrazitelný stromovou strukturou s originálním systémem v kořeni.

4.2 Aplikace metod

a) objektově-orientovaná analýza a design (OOAD) — Přístup „zdola nahoru“, jehož základní myšlenkou je zobecňování. Začíná se z konkrétní části systému, která se analyzuje a popíše. V dalších částech se pak hledají společné rysy, z nichž se odvozují obecné zákonitosti. Výhodou je možnost rychlého paralelního vývoje prototypového software; nevýhodou je nedostatek ověřených metodických postupů a hlubší zkušenosti s rozsáhlejšími systémy. Typickým příkladem je UML (Unified Modelling Language).

P.S. OOAD na naší katedře téměř nikdo nezná, a když, tak jenom povrchně. Trochu se o to zajímá Vlček, Pěchouček, Lažanský. V praxi to používá snad jen externista Radek Mařík (co přednáší KTS). Ale OOAD pomocí UML má velkou perspektivu — viz kapitolu 5.1.

b) strukturální analýza a design — Klasický postup „shora dolů“. Chápe systém jako celek a postupně ho dekomponuje na jednodušší části. Je metodicky velmi dobře zvládnut. Nevýhodou je absence možnosti předčasně ukončit analýzu a implementovat konkrétní část systému. Typickým příkladem je metoda SSADM (Structured Systems Analysis and Design Method) — viz kap. 5.2.

4.3 Nástroje analýzy

Největší část systémového analytika spočívá ve tvorbě různých modelů systému. Jsou různé pohledy na systém (a tedy i modely):

4.3.1 Funkční model systému — diagramy datových toků

Je nazýván různými názvy — procesní model, digram toku práce, funkční model, bublinový diagram, diagram datových toků (Data Flow Diagram — DFD).

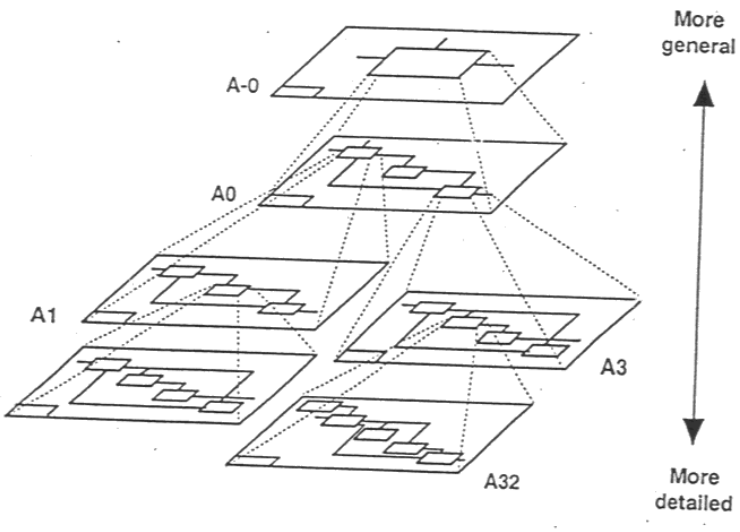
Nejvíce se používá asi označení DFD. DFD je síťovou reprezentací systému. Systém může být automatizovaný, manuální nebo smíšený. DFD znázorňuje systém pomocí jeho komponent a určuje rozhraní mezi komponentami.

DFD se skládají ze 4 komponent:

- datové toky — popisují pohyb dat
- procesy — transformují vstupy na výstupy
- datové paměti — kolekce datových skupin v klidu
- terminátory — zdroje a příjemce dat v okolí systému, např. osoby nebo spolupracující systémy

Jsou různé druhy notací, neznámější jsou Yourdon/DeMarco, Gane/Sarson a SSADM.

Reálný systém nelze popsat jediným DFD modelem. Proto se používá stromová struktura, v jejímž koření je tzv. kontextový diagram. Viz obr. 6.



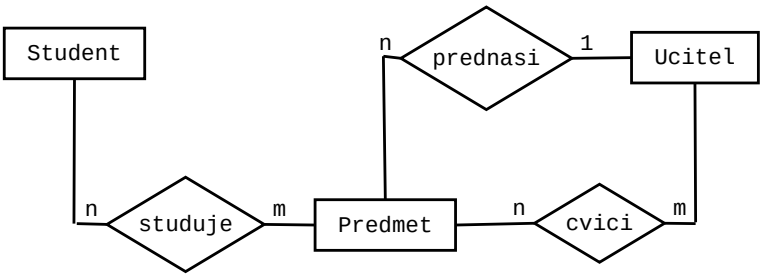
Obrázek 6: Víceúrovňová hierarchie DFD

4.3.2 Modely vnějšího chování

Jsou to např. již zmíněný *kontextový diagram* a také *seznam událostí*, které působí na systém z jeho okolí.

4.3.3 Datové modely — ER diagramy

Tyto diagramy jsou jádrem pro databázové systémy. Určují vztahy mezi objekty. Např. pro modelování informačního systému lze použít vztahy: „učitel přednáší předmět(y)“, „student studuje předmět“, ard. viz obr. 7.



Obrázek 7: ER diagram studia

Na tomto obrázku je také vidět významný atribut každého vztahu — arita. Charakterizuje počet entit, vstupujících do každého vztahu.

4.3.4 Stavové diagramy

Stavové diagramy modelují časově závislé chování systému. Ty nebudu ani popisovat, ty snad všichni známe.

5 Metody návrhu a tvorby systémů

Zde si ukážeme nějkaé příklady metod a metodik návrhu systémů. Zmíníme jednu objektovou — UML, a jednu strukturovanou — SSADM.

5.1 UML

UML (Unified Modeling Language) je průmyslový standard, je to jazyk pro specifikaci, vizualizaci, konstrukci a dokumentování nejen softwarových systémů, ale i obchodních modelů a i nesoftwarových systémů. Zjednodušuje komplexní proces návrhu softwaru, např. vytvořením „kostry“ programu.

Je to poměrně nová věc. Společnou snahou hlavně firmy Rational Software a jejích partnerů vznikla v roce 1997 verze UML 1.0, která se poslala OMG (Object Management Group) pro standardizaci. V současnosti je verze 1.3.

5.1.1 Různé pohledy

Principy UML lze nastínit několika body:

- Na každý složitý problém je vhodné se dívat z více úhlů pohledu. Jeden pohled nestačí.
- Každý model může býti vyjádřen s různou přesností.
- Nejlepší modely jsou pak spojeny s realitou.

Proto UML definuje následující typy diagramů, které umožňují různé úhly pohledu.

- diagram scénářů [use case diagram]
- diagram tříd [class diagram]
- diagramy chování
  - stavový diagram [statechart diagram]
  - diagram aktivit [activity diagram]
  - interakční diagramy
    - sekvenční diagram [sequence diagram]
    - diagram spolupráce [collaboration diagram]
- implementační diagramy
  - diagram komponent [component diagram]
  - diagram zaměstnanosti [deployment diagram]

Obrázky jsou např. na internetu [1].

5.2 SSADM

Tak toto je velmi striktní metodika návrhu, která de fakto využívá výše zmíněné věci. Moc ji neznám, navíc k ní nejsou volně dostupné softwarové prostředky (aspoň co já vím).

Reference

[1] Dobiáš, L.: referát z předmětu CSD na téma *Modelování pomocí UML (programem Rational Rose)*. Praha 1999. Dostupný na URL: `http://cs.felk.cvut.cz/~xdobiasl/`.

[2] Mařík, R.: „slajdy“ z přednášek předmětu *KTS — Kvalita a testování softwaru*, dostupné na URL: `http://labe.felk.cvut.cz/~marikr/teaching/indexSQT.html`. Psány anglicky. Praha 1999.

[3] Richta K., Sochor J.: *Softwarové inženýrství I*. Skripta. Vydavatelství ČVUT, Praha 1988. ISBN 80-01-01428-2.

[4] Sborník 4. ročníku celostátní konference Objekty ’99, článek Daniel Štourač: *Příklad tvorby informačního systému s použitím UML*. Praha 1999. Je k dispozici v knihovně K333, knihovní číslo 768.

[5] Vlček, T.: „slajdy“ z přednášek předmětu *Počítačem podporovaná výroba* dostupné na URL: `ftp://labe.felk.cvut.cz/pub/vyuka/33PPV/`, hlavně soubory `ppv3-4.pdf` a ppv5.pdf. Praha 1999.

[6] Přednášky a skripta z předmětu *Počítače a programování*, část o Objektovém programování, autora nevím, přednášel Ivan Jelínek. Praha, ČVUT FEL asi 1995.