NOVÉ POHLEDY NA ŽIVOTNÍ CYKLUS TVORBY SOFTWARE Z HLEDISKA JAKOSTI APLIKACÍ AUTOMATICKÉHO ŘÍZENÍ

Branislav Lacko

Vysoké učení technické v Brně, FSI, Brno

Abstrakt
Příspěvek popisuje různé modely životního cyklu vývoje software. Analyzuje rozdíly mezi přímočarým vodopádovým modelem a modelem s kontinuálního textování s neustálými zpětnými návraty.

1. Úvod

Neustálý vzrůst počtu počítačů a dalších programovatelných mikroprocesorových systémů přináší s sebou zvýšené požadavky na tvorbu programového vybavení všeho druhu. Zdá se, že se prostřednictvím moderních programovacích jazyků podařilo částečně překonat softwarovou krizi, o které se tolik hovořilo začátkem osmdesátých let minulého století. Objevil se však jiný kritický faktor - čas. Termíny na vytvoření nového potřebného software jsou stále kratší. Navíc je potřeba dnes více přihlížet k požadavku dosahování vysoké jakosti programového vybavení. Proto softwarové inženýrství neustále hledá cesty, jak těmto imperativům současné doby úspěšně vyhovět.

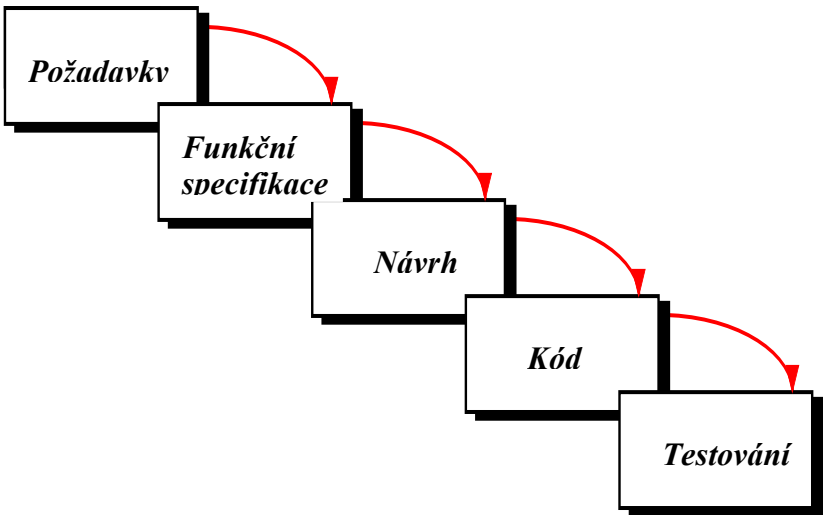
Kromě jiných možností se hledá řešení i v samotné organizaci základního přístupu k tvorbě programů. Diskuse, která poukazovala např. na nutnost změny organizaci při tvorbě software v okamžiku přechodu na objektově orientovanou technologii proběhla na semináři TS 2000. [7]

Příspěvek si klade za cíl ukázat jak se mění názor na řešení základního postupu tvorby software. Základní představy o tvorbě software bývají popisovány prostřednictvím modelů životního cyklu software, což je soustava procesů, činností a úloh zahrnutých do akvizice, vývoje, provozování a údržby programových produktů [5]

2. Přehled modelů životního cyklu

Životní cykly, které popisují základní představu o tvorbě softwaru, nazýváme často konceptuálními modely tvorby software. Zabývá se jimi softwarové inženýrství. [4,7] Rozeznáváme jich celou řadu:

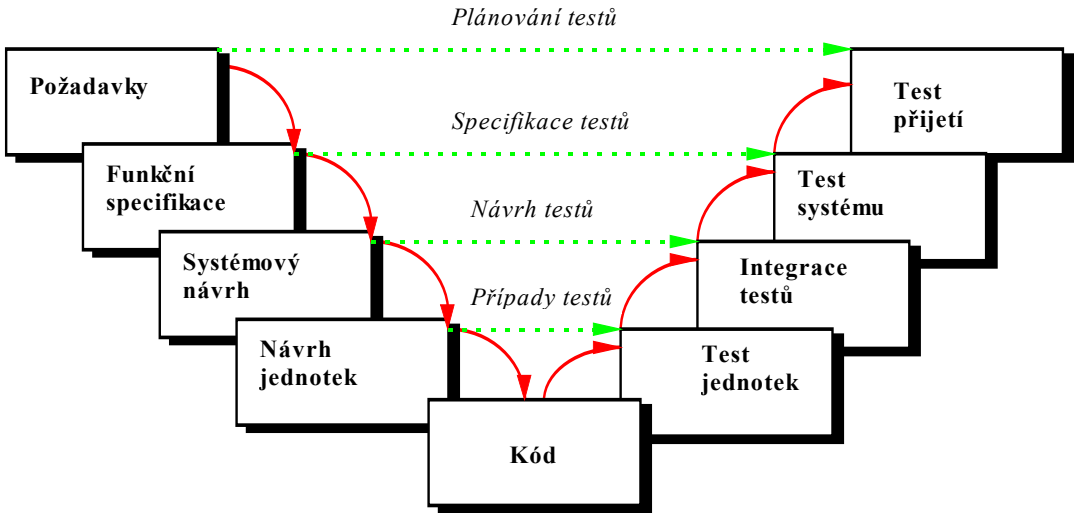
- **Vodopádový životní cyklus (The Waterfall Life - cycle):**



Obrázek: Vodopádový životní cyklus

Vodopádový model neodráží známou skutečnost potřeby „zpětných kroků“ při tvorbě software a nutnost důkladného prověřování výsledků každé etapy.

- **V - životní cyklus (The V Life Cycle)**

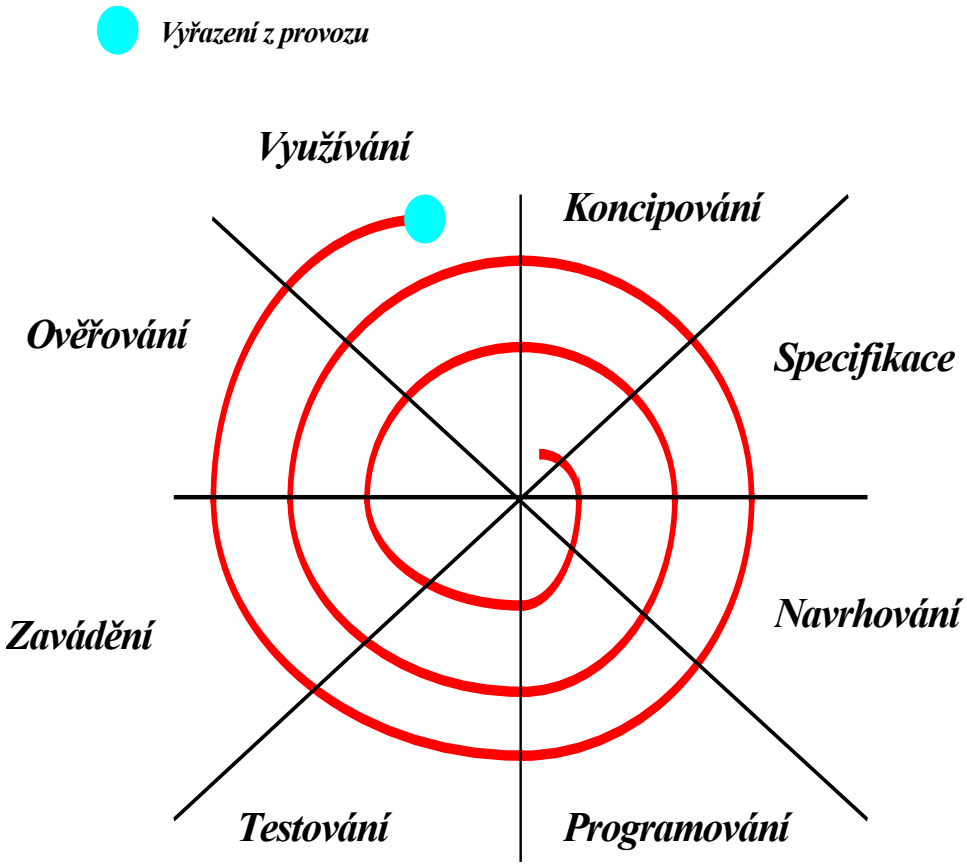


Obrázek: V - životní cyklus

Tento model vychází z konceptu potřeby neustálého testování, aby se dosáhla vysoká jakost software. Ukazuje nutnost plánovat testy současně se vznikem požadavků na ověřování jednotlivých kroku

V poslední době se v důsledku použití prototypovacích nástrojů zdůrazňuje iterační postup při tvorbě programového vybavení, jehož schématické znázornění je na následujícím obrázku. Chce se zdůraznit, že vývoj probíhá postupně,stále na kvantitativně širší a kvalitativně vyšší úrovni, a že je omezen okamžikem, kdy se produkt z různých důvodů (zásadní změna technického vybavení, změna uživatelského přístupu apod.) vyřadí z používání. V anglosaské literatuře bývá označován pojmem ITERATION LIFE CYCLE nebo PROTOTYPING LIFE CYCLE. Ten lépe vystihuje okamžik zahájení tvorby software, tvorbu jednotlivých verzí programového produktu a zachycuje i okamžik ukončené používání programového produktu. Navíc vyjadřuje skutečnost zvyšování kvalitativní úrovně v průběhu jednotlivých cyků.

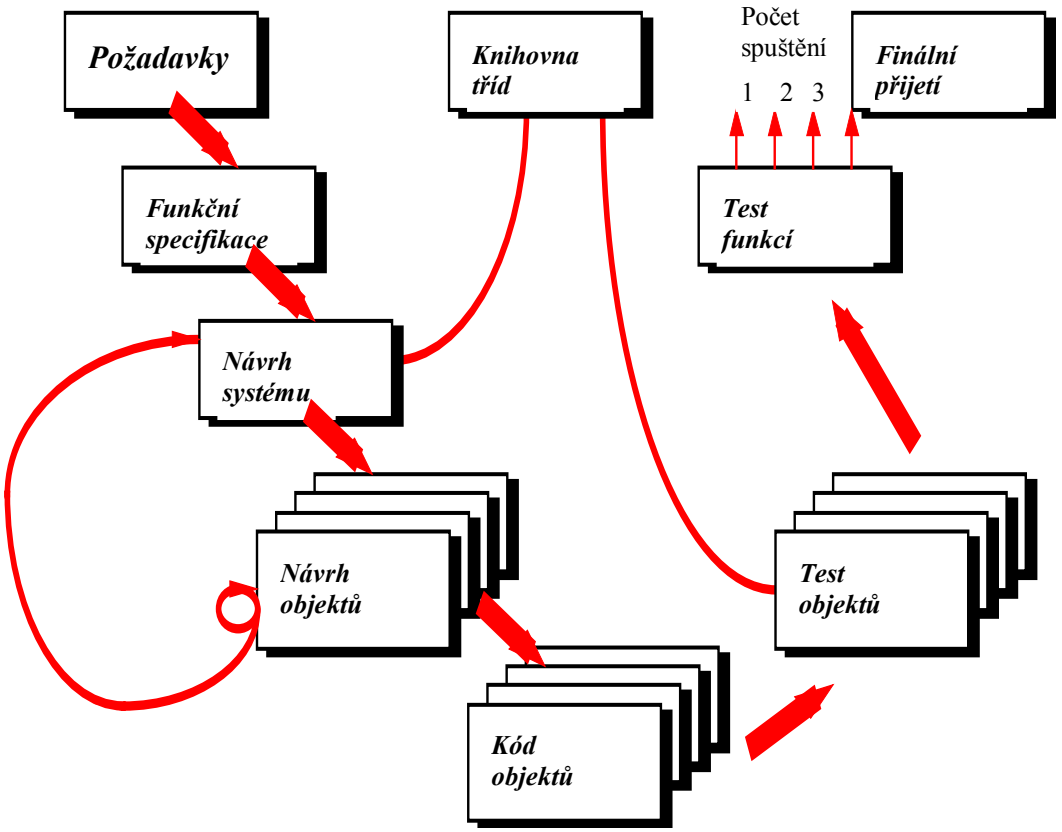
■ **Iterační životní cyklus (Iteration Life Cycle)**



Obrázek: Iterační životní cyklus

Dalším možným typem životního cyklu software, jehož použití bude v budoucnu stále narůstat na významu je „ Objektově orientovaný životní cyklus“, který odráží objektově orientovaný přístup ke tvorbě software. Významnou součástí komponent modelu je knihovna tříd, která slouží jako zásobník programových prvků pro nově vytvářená řešení.

■ **Objektově orientovaný životní cyklus (An Object Oriented Life Cycle)**

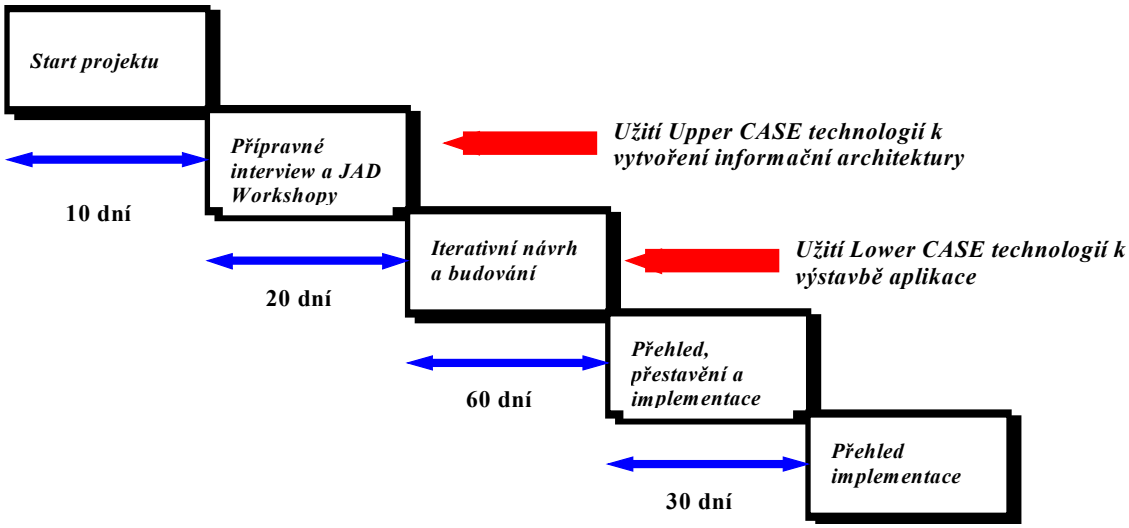


Obrázek: Objektově orientovaný životní cyklus

Velmi moderním trendem vývoje aplikačního softwaru je vývoj se zvýšenou společnou účastí budoucích uživatelů a s využitím myšlenky rychlého vytvoření základního jádra aplikace, které se postupně rozšiřuje se současným využíváním prvotně vytvořených programových funkcí.

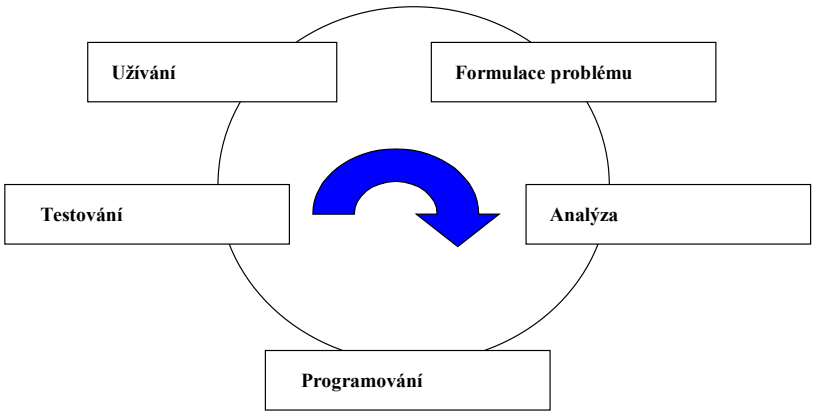
Tímto modelem je tzv. Rychlý vývoj aplikace (Rapid Applications Development), který reaguje na požadavek zkrácení doby vývoje software..

- **Rychlý vývoj aplikace (Rapid Applications Development)**



Některé publikace uvádějí tzv. Kruhový životní cyklus tvorby software, který vystihuje cyklickou tvorbu jednotlivých verzí programových produktů, ale ukazuje počáteční moment

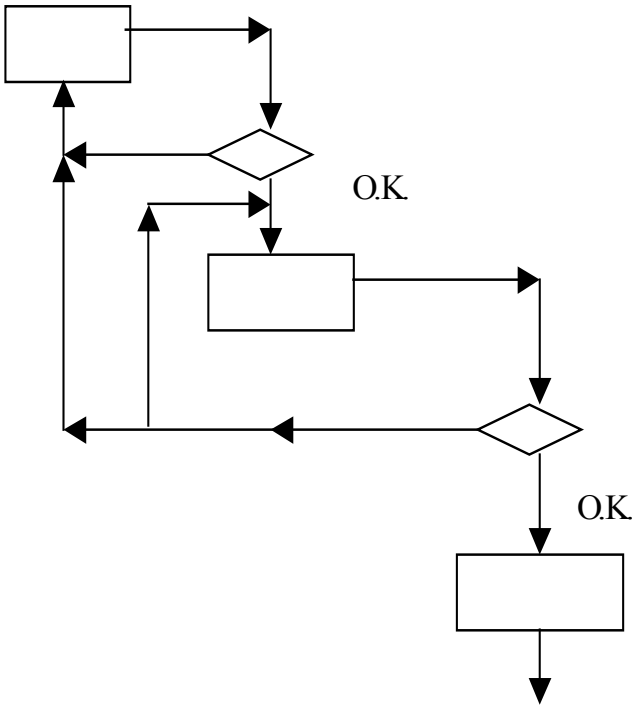
- **Kruhový životní cyklus**



vzniku produktu ani konec jeho vytvářen:

Problematiku návratových kroků na základě vyhodnocení výsledků po ukončení každé fáze vystihuje nejlépe model životního cyklu kontinuálního testování podle japonského profesora Fujiho.

- **Model kontinuálního testování**



Poznamenejme, že norma ISO 12 207 nenařizuje dělení životního cyklu do určitých fází. Doporučuje dělení na následující procesy:

- Proces akvizice software
- Proces vývoje software
- Proces provozu a údržby software

Pro tyto základní procesy doporučuje norma jejich členění na subprocesy.V konkrétních případech je však možno členit životní cyklus podle specifických potřeb. Pro potřeby příspěvku byl životní cyklus tvorby software rozdělen na základní kroky: formulace zadání/problému-analýza-programování-testování- užívání.

3. Výběr dvou modelových typů

Postup a plánování činností, které je potřeba vykonat při tvorbě software, je ovlivňován filosofií, jak činnosti vykonávat. Pro srovnání si zvolíme vodopádový životní cyklus jako model, který představuje typickou variantu přímočarého postupu bez vratných kroků při tvorbě software (model A) a životní cyklus kontinuálního testování, který představuje variantu nepočetnějších možností vratných kroků (model B)

Model A představuje filosofii, založenou na zajišťování jakosti software na principu „a priori“. Zásadou je chybám a problémům předcházet. Děje se tak realizací různých opatření, z nichž nejdůležitější jsou:

- Promyšlený, předem připravený metodický postup
- Odstranění nebo minimalizace vlivu faktorů, které mohou být příčinou chyb
- Nalezení chyby co možná nejdříve, kdy její odstranění je často možno provést ne operativní úrovni téměř ihned s minimálními náklady
- Neustálá, pokud možno automatizovaná, kontrola výsledků všech úkonů a činností
- Předcházet chybám na základě zjištěných negativních trendů
- Zjednodušení a racionalizace všech prováděných činností
- Zajištění vysoké kvalifikace pracovníků, kteří činnosti provádějí
- Vysoká motivace a zainteresovanost všech pracovníků na jakosti softwarového produktu

Výhodou je skutečnost, že při promyšlené realizaci výše uvedených zásad lze dosáhnout téměř přímočarého postupu a jakostního výstupního produktu.
Logickou nevýhodou jsou dodatečné náklady, spojené se zajištěním výše uvedených zásad a prodloužení celého procesu o kontrolní a jiné činnosti.

Model B je založen na filosofii „a posteriori“, kdy na základě poznání výsledků vracíme se snažíme opravit produkt opakováním jednotlivých činností a postupným odstraňováním zjištěných chyb zlepšovat jakost produktu. Tento postup se snaží minimalizovat náklady na kontrolu tím, že testuje až finální produkt nebo zařazuje jen ojedinělé testy ve vybraných okamžicích, kdy jsou v procesu vývoje ukončeny některé fáze životního cyklu. Většinou se předpokládá, že minimální kontroly postačí a že dílčí ušetřené náklady a dílčí ušetřený čas se projeví v celkových ušetřených nákladech a zkrácení celkové doby vývoje programového produktu.

V kapitole č. 3 bylo uvedeno sedm různých modelů. Můžeme si povšimnout, že kromě vodopádového modelu, všechny ostatní řeší určitým způsobem návratové kroky při tvorbě softwaru. Přitom model kontinuálního testování představuje z nich principiálně nejsložitější

variantu, z hlediska návratů. Proto byl vybrán jako jejich typický představitel a je srovnáván s přímočarým postupem vodopádového modelu.

4. Srovnání vybraných modelů

Většina softwarových firem dává přednost modelům se zpětnými kroky. Řada z nich používá především buď přímo model kontinuálního testování nebo vlastní firemní model, který z něho vychází. Přesto, že toto tvrzení nelze podložit výsledky exaktního průzkumu, potvrzuje to existence miliónů programových produktů, které jsou na trh dodány s chybami, některé i s fatálními chybami, a to často po mnohonásobném odkládání termínu uvedení na trh!. Za fatální chybu u operačního systému je potřeba považovat případ, kdy operační systém se zhroutí uprostřed běžné práce bez nějaké mimořádné příčiny. To se velmi často stává i některým verzím nejrozšířenějšího operačního systému WINDOWS, jehož počet kopií dosahuje dokonce několika desítek miliónů kusů.

Manažeři softwarových firem uvádějí obvykle následující důvody, proč preferují model typu B s využitím zpětných kroků:

- Výrazně menší organizační úsilí. Prakticky je ponecháno vše samovolnému průběhu, který není nutno vynucovat, tedy ani financovat (O to více stojí náprava důsledků, které jsou způsobeny nedodržením termínů, překročením nákladů, špatnou kvalitou produktu a náklady na různé vícepráce, atd.)
- Při tvorbě software nutně dochází k chybám. S tím se nedá nic dělat! Tak proč se zbytečně namáhat! (Resignace a akceptace současného nepříznivého stavu)
- Není přece vyloučeno, že pokud nám bude přát štěstí, může celý průběh být velmi krátký. (Neoprávněný optimismus a hazardní spoléhání na šťastnou náhodu)
- Všechny ty "vědecké modely" vyžadují samé kontroly a spoustu všelijakých složitých činností, takže prodlužují délku projektu. (Nikdo nevyhodnotí, o kolik se skutečně prodloužil projekt, který trval nakonec velmi dlouho v důsledku různých zmatků a chyb)

Dalším nedativním důsledkem aplikace modelu typu B je nemožnost plánovat celý proces tvorby software a to jak z hlediska časových termínů, tak z hlediska nákladů a potřeby zdrojů. Ve svém důsledku to znemožňuje jakostní řízení celého projektu!

Model typu A je založen na přímočarém postupu s jasným vymezením plánovaných termínů.

Pokud hovoříme o skutečnosti, že v modelu typu A nejsou zpětné kroky, je potřeba toto tvrzení vysvětlit. V první řadě se jedná o skutečné dodržení přímočarého postupu, kterého se dosahuje promyšleným metodickým postupem a využitím počítačové podpory produkty CASE. Je však nutno upozornit na to, že do určité míry je přímočarý postup zdánlivý. Každá logicky ucelená skupina činností je ukončena náročnými testy SW produktu a po testech je vždy plánována činnost resp. činnosti, pro korekci zjištěných nedostatků. Významným faktem je, že potřebné korekce nejsou rozsáhlé, je jich málo a po jejich provedení má mezivýsledný SW produkt potřebnou jakost. Rovněž na začátku významných fází jsou zařazovány testy, zda není potřeba provést některé korekce SW produktu. Pokud ano, provedou se potřebné korekční činnosti. Jestliže v obou případech se zjistí, že není potřeba korekce provádět, dostává se projekt do časového předstihu. Je to opačný přístup než v případě modelu B, kde se většinou provede optimistický časový odhad, aby se zjistilo, že je potřeba přece jen provést řadu opakovaných činností, které vedou k prodloužení projektu a k časovému skluzu.

Poznamenejme, že pokud se v případě modelu typu B se vyskytnou požadavky na významné změny nebo na jejich velké množství, znásobí se počet potřebných zpětných kroků (někdy až tak výrazně, že to vede ke značnému odsunutí termínu ukončení projektu. V případě A jsou obvykle stanoveny přísná pravidla, limitující počet a rozsah změn. Pokud je potřeba provést velké změny nebo jejich vysoký počet, zásadně se přehodnocuje celý projekt přeplánováním z hlediska nákladů a termínů nebo se dokonce původní projekt zastaví a odstartuje se nový projekt, akceptující požadovaný rozsah změn.

V poslední době se často diskutuje o metodě SIX SIGMA [6]. Níže uvedená tabulka ukazuje jak klesá počet nehod s dosaženou úrovní sigma. Dále jsou uvedeny souhrnné náklady na jakost i neshodné výrobky.

Úrovně v metodě SIX SIGMA		
Úroveň sigma	**DPMO – počet neshod na milion příležitostí**	**Náklady na jakost**
2.	308 537 (konkurence neschopná společnost)	Neřízené – jsou výsledkem náhody
3.	66 807	25-40% z prodejní ceny
4.	6210 (Průměrná společnost)	15-25% z prodejní ceny
5.	233	5-15% z prodejní ceny
6.	3,4 (World class)	<1% z prodejní ceny
Každý posun o jednu úroveň sigma znamená 10% zvýšení čistého příjmu		

Metoda SIX SIGMA je založena na přímočarém postupu k jakostnímu výrobku a ukazuje jednak skutečnou možnost takové cesty, jednak získané výhody.

Kromě metody SIX SIGMA i práce Software Engineering Institute of Carnergi Mellon University v Pittsburgu ukázaly na význam procesů při tvorbě softwaru.[8]. Pět úrovní modelu, označovaného zkratkou SW-CMM (Software Capability Maturity Model) ukazuje na důležitost procesního řízení:

Číslo úrovně	**Označení úrovně**	**Charakteristika procesů při tvorbě software**
1.	Initial	Nahodilé procesy
2.	Repeatable	Stabilizované repetiční procesy
3.	Defined	Popsané a definované procesy
4.	Managed	Řízené procesy
5.	Optimizing	Optimalizované procesy

5. Závěr

Problematika jakostního řízení projektů, jejichž obsahem je tvorba programového vybavení pro aplikace automatického řízení je dnes velmi aktuální. Roste počet mikroprocesorů, které jsou zabudovávány do různých výrobků. V řadě případů dochází k nepříjemné skutečnosti, když se vývoj programových prostředků opožďuje za vývojem technických prostředků příslušného výrobku. Tvorba potřebného programového vybavení se pak stává rizikovým faktorem, který je potřeba velmi důkladně analyzovat a připravit opatření pro snížení rizika [4].

Analýze postupu při tvorbě softwaru byla věnována pozornost i na konferencích Tvorba softwaru v letech 2000 a 2002. [1,2]. Pro konferenci TS 2004 je připravován příspěvek, který by měl prostřednictvím modelováním a simulace umožnit srovnání obou modelů prostřednictvím kvantitativních parametrů. Výsledky by měly přesvědči ty, kteří ještě pochybují o výhodách a nutnosti zvýšení jakosti při řízení tak významných projektů, které jsou představovány případy tvorby programů pro automatické řízení.

Požadavky na jakost programů se budou zřejmě neustále zvyšovat z mnoha důvodů. Mikroprocesorové systémy se budou používat ve stále větším počtu i tam, kde by selhání jejich programového vybavení mohlo přivodit katastrofální následky. Do programového vybavení jsou investovány každým rokem obrovské investice a je potřeba zabránit jejich znehodnocení. Zákazníci stále rostoucího trhu software chtějí být chráněni před nejakostními programy. Nakonec je nutno připomenout snahu zemí Evropské unie prosadit filosofii vysoké jakosti prostřednictvím revidované řady norem ISO 9000:2000.[9]

Cesty, které naznačily takové přístupy jako SW Capability Maturity Model, Six-Sigma a další ukazují, že požadavky vysoké jakosti, které prosazuje soubor norem ISO 9000:2000 je možno splnit i v softwarových firmách, které produkují tak specifický produkt své práce, jakým je programové vybavení počítačů

Správné vyřešení základních otázek kolem životního cyklu tvorby software je základem pro řešení celé řady dalších problémů při zvyšování jakosti nejen řídicích programů ale software obecně.

Literatura:

1. Kubis, J.: Existenčné riziká projektu a k nemu viazaných subjektov, In: Sborník konference Tvorba softwaru 2002. VŠB-TU Ostrava 2002, str.109-116
2. Kubiš, J.: Plánovaie a zavádzanie informačných systémov na báze normatívov, In: Sborník konference Tvorba softwaru 2000. VŠB-TU Ostrava 2000, str. 66-75
3. Lacko, B.: Aplikace metody RIPRAN v softwarovém inženýrství. In: Sborník konference Tvorba softwaru 2001. VŠB-TU Ostrava 2001, str. 97-103
4. Král, J-Demner,J: Softwarové inženýrství. Academia Praha 1991
5. ČSN/ISO 12207 Informační technologie. Procesy v životním cyklu software
6. Pande,P.-Neuman, R.- Cavanagh,R.: Zavádíme metodu Six Sigma. TwinsCom Brno 2002
7. Zyka, V.: Řízení objektově orientované analýzy a designu. In: Sborník konference Tvorba softwaru 2000. VŠB-TU Ostrava 2000, str. 195-204
8. Paulk,C.,M.: A Comparison of ISO 9001 and the Capability Matutity Model for Software. Software Engineering Institute. CMU/SEI-94-TR-12, July 1994
9. Conti,T.: Návrh Evropské vize jakosti. In: Sborník mezinárodní konference „Kvalita v Evropě – základ pro 21.století“.Česká společnost pro jakost, Praha 2000, str. 7-15

Příspěvek byl zpracován v rámci výzkumného záměru MŠMT J22/98:26000013 “Automatizace technologií a výrobních procesů“