VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

ZVLÁŠTNOSTI SOFTWARE Z HLEDISKA KVALITY

SOFTWARE SPECIALTIES IN TERMS OF QUALITY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

Marie Zlatníčková

VEDOUCÍ PRÁCE
SUPERVISOR

doc. Ing. Branislav Lacko, CSc.

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání bakalářské práce

Ústav:	Ústav automatizace a informatiky
Studentka:	**Marie Zlatníčková**
Studijní program:	Strojírenství
Studijní obor:	Aplikovaná informatika a řízení
Vedoucí práce:	**doc. Ing. Branislav Lacko, CSc.**
Akademický rok:	2018/19

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma bakalářské práce:

Zvláštnosti software z hlediska kvality

Stručná charakteristika problematiky úkolu:

Zpracovat přehled zvláštností, kterým se liší software od hmotných produktů z hlediska potřeb měření a řízení kvality programových produktů.
Analyzovat seznam a obsah norem ISO, které problematiku kvality software popisují.

Cíle bakalářské práce:

Specifikujte požadavky na jakost softwaru.
Sestavte přehled zvláštností software.
Uveďte důvody pro zvýšení důrazu na kvalitu softwaru v současné době.

Seznam doporučené literatury:

PATTON R.: Testování software. Computer Press 2002 Praha.

ISO/IEC 25000:2014 Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- Guide to SQuaRE.

BECK, K: Programování řízené testy. Grada 2004 Praha.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2018/19

V Brně, dne

L. S.

doc. Ing. Radomil Matoušek, Ph.D. ředitel ústavu	doc. Ing. Jaroslav Katolický, Ph.D. děkan fakulty

ABSTRAKT

Tato bakalářská práce se zabývá popisem softwarových specifikací a požadavků na kvalitu softwaru, popisem procesů testování softwaru za účelem prokazování jeho kvality a metodám zvyšování kvality.

ABSTRACT

The Bachelor's thesis deals with describing software specifications and software quality requirements, software testing for proving quality and applying software quality improving methods.

KLÍČOVÁ SLOVA

Kvalita softwaru, norma ISO 25000, specifika softwaru, model SQUARE

KEYWORDS

Software quality, standard ISO 25000, software specifications, SQUARE model

BIBLIOGRAFICKÁ CITACE

ZLATNÍČKOVÁ, M. *Zvláštnosti software z hlediska kvality*. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2019. 63 s. Vedoucí bakalářské práce doc. Ing. Branislav Lacko, CSc.

PODĚKOVÁNÍ

Tímto bych ráda poděkovala doc. Ing. Branislavu Lackovi, CSc. za cenné připomínky, rady, ochotu a pomoc při vytváření této bakalářské práce. Ráda bych také poděkovala své rodině za podporu a pomoc v průběhu celého studia.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že tato práce je mým původním dílem, zpracovala jsem ji samostatně pod vedením doc. Ing. Branislava Lacka, CSc. a s použitím literatury uvedené v seznamu literatury.

V Brně dne 24.5.2019 ……

Marie Zlatníčková

OBSAH

1 ÚVOD

Kvalita spotřebního zboží a služeb v dnešní době je velmi důležitá. Kvalitou označujeme dobrou úroveň sledovaných vlastností daných výrobků a služeb nebo do jaké míry odpovídá jakost služby nebo výrobku požadavkům zákazníka či standardům daným normami. Vytváření kvalitních výrobků (např. softwaru) a služeb je žádoucí, jak z hlediska poptávky, tak z hlediska nabídky.

Jedním z problémů kvality je prokázat, že produkt skutečně kvalitní je. K prokazování kvality se využívá testování. Cílem je odstranit chyby v softwaru vzniklé při jeho realizaci. Chybami je zde označován nesoulad mezi požadavky zákazníka, požadavky dané normami a předpisy a skutečnou realizací daného softwaru. Je velmi těžké nebo spíše nemožné odstranit všechny chyby, protože z různých hledisek mohou být různé skutečnosti považovány za chybu nebo za normální chod programu. Testováním se snažíme snížit chybovost softwaru na minimum. Čím menší chybovost tím je software pokládán za kvalitnější.

V posledních letech se i v České republice prosazuje koncept Průmysl 4.0, což je snaha o nahrazení nebo minimalizování lidské činnosti ve výrobním procesu, přechod na plně automatizované řídící systémy, robotizace, digitalizace procesů, komunikace jednotlivých systémů pomocí internetu věcí apod.[1] Těmito prostředky dochází ke snížení nákladů na výrobu, zvyšuje se efektivita výrobních procesů a vznikají nová pracovní místa. Tam, kde jsou všechny nebo většina prostředků plně ovládané na dálku přes internet, tam je zapotřebí zajistit bezpečí přenášených dat a zařízení která je přenáší před cílenými útoky,[2] protože při cíleném útoku na určitá zařízení ve výrobním procesu může dojít např. k vyřazení výroby a k následným velkým ztrátám zisku. Kvalita softwaru v konceptu Průmysl 4.0 představují jeden z kritických faktorů úspěchu jeho praktického použití.

Cílem této bakalářské práce je tedy charakterizovat požadavky na kvalitu softwaru z hlediska norem popisujících jakost a kvalitu zboží a služeb, popsat problematické zvláštnosti softwaru, testování a možná řešení za účelem zvýšení jeho kvality.

Na závěr zhodnotím dosažení cílů této bakalářské práce, současný stav kvality softwaru a možná řešení vedoucí ke zvýšení jeho kvality.

2 EVROPSKÁ SNAHA O KVALITU

2.1 Charakteristika řady norem ISO 9000

Normy ISO řady 9000 se zabývají popisem požadavků na kvalitu výrobků, služeb a na kvalitu zabezpečení.[3] Mají univerzální použitelnost pro všechna odvětví, ale jejich charakter je pouze doporučující.[5] Historie ISO 9000 je svým způsobem soubor norem popisujících kvalitu a její řízení z různých hledisek. Je tvořena čtyřmi základními normami:

- ISO 9000:2005 Systémy managementu kvality - Základní principy a slovník: definuje základní principy managementu jakosti a obsahuje výklad pojmů použitý v dalších normách.[4, 5]
- ISO 9001:2000 Systémy managementu kvality - Požadavky: má snahu být celosvětově uznávaným kriteriálním modelem pro certifikaci systémů managementu jakosti.[6, 5]
- ISO 9004:2000 Systémy managementu kvality - Směrnice pro zvyšování výkonnosti: je určena pro vnitřní aplikaci v organizacích a k jejímu hlavnímu cíli patří zavádění principů managementu jakosti do praxe, v České republice je tato norma zřídka kdy využívána, protože nepatří mezi kritéria pro certifikaci.[6, 5]
- ISO 19011:2002 Směrnice pro auditování systémů managementů jakosti a systémů environmentálního managementu.[6, 5]

2.2 Norma ISO 9001

Tato norma se zabývá snahou o docílení jednotného systému řízení kvality v celosvětovém měřítku. Norma ISO 9001 vznikla jako kompromis minimálních možných požadavků na kvalitu, požadovaných jednotlivými zúčastněnými stranami.[7]

Princip fungování normy: norma stanovuje zásadu, podle které vedení firmy nastaví své plány a cíle v oblasti produkce a ty jsou poté postupně pomocí nastavených procesů realizovány. V průběhu realizace je měřena a monitorována účinnost výrobních procesů, aby společnost mohla dostatečně včas zareagovat na změny na trhu.[7]

ISO 9001:2015 byla vytvořena s cílem reagovat na změny v prostředí, jak podnikatelském tak společenském a úspěšně využívat zkušenosti z dlouholeté aplikace normy verze ISO 9001:2000.[7]

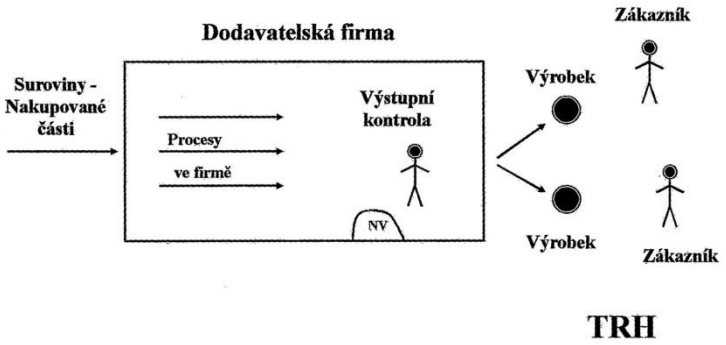
Mezi nejvýznamnější cíle znovelizované normy patří: udržení významu a hodnot pro podniky a jejich klienty, větší kompatibilita s ostatními normami systému řízení kvality, lepší návaznost na všeobecný systém řízení firmy a následné lepší praktické využití této skutečnosti v praxi, schopnost umět zareagovat na rychlejší změny tržního

prostředí, propojení s marketingem a následná lepší schopnost uspokojovat potřeby a přání zákazníků, zajištění stability normy na dalších 10 let.[7]

Významným novým požadavkem je explicitně vyžadované zvažování rizik jejich analýzou a plánem opatření na jejich snížení z hlediska nedodržení požadované kvality.

2.3 Evropské procesní pojetí kvality

V Evropě je prosazován přístup, který popisuje kvalitu jako výsledek určitého procesu. Proces je určitý postup, při kterém zpracováním vstupů získáme výstupy. Všeobecné výrobní procesy ve firmě asi nejlépe vystihuje obr. 1.



Obr. 1: Výrobní procesy, náklady na neshodné výrobky.[8]

Firma nakupuje suroviny na výrobu, zpracuje je výrobními procesy a výsledkem jsou výrobky. Před předáním výrobků zákazníkům, jsou všechny výrobky zkontrolovány, zda vyhovují požadavkům a pokud vyhovují, jsou předány zákazníkovi. Neshodné výrobky (v obrázku NV) jsou vyřazeny a do prodeje už nejdou, tyto výrobky způsobují firmě ekonomické ztráty, pokud se jejich množství zvyšuje, pak dochází ke zvyšování nákladů na výrobu.[8]

Neshodné výrobky jsou ty, které neodpovídají požadavkům zákazníka nebo se nevejdou do tolerance. Tolerance je rozmezí určitých parametrů, kdy jsou ještě výrobky považovány za přijatelné. Neshodnost výrobků může být způsobena např. špatně kalibrovanými přístroji, špatným nastavením použitého softwaru, lidskou chybou apod.[8, 9]

Abychom mohli vyrábět kvalitní produkty, je důležité si uvědomit, jaké jsou naše celkové požadavky na produkt a jeho kvalitu, tato tématika je lépe shrnuta v [10] a podrobněji se jí zabývám v průběhu celé práce.

Soubor norem ISO 9000 je reakcí Evropy na vysokou kvalitu v Japonsku a v USA [11]. V USA je prosazován jiný model pojetí kvality softwaru než v Evropě, tento model se nazývá CMMI (Capability Maturity Model Integration) a jeho původní verze CMM (Capability Maturity Model) byla vytvořena na univerzitě v Pittsburgu.[12, 13]

3 ZVLÁŠTNOSTI SOFTWARE Z HLEDISKA KVALITY

3.1 Problematické zvláštnosti software

Software byl už od počátku své existence svým způsobem problémové zboží. Jelikož je software nehmotný produkt, tak není jednoduché změřit jeho parametry a kvalitu. Hlavním znakem hodnocení kvality softwaru je neobjektivnost. Kvalita softwaru je vnímána každým uživatelem jinak. Proto je velmi obtížné přesně definovat jeho kvalitu.

První vytvořený software se datuje už do 60. let minulého století, ale už od prvopočátku své existence se potýká s mnoha problémy. Jedním z hlavních problémů je přesná definice toho, co software má a nemá dělat, protože zákazníci jsou v mnoha případech lidé, kteří většinou vědí velmi málo o celém procesu vývoje softwaru a tak se může stát, že budou mít nereálné nebo velmi těžko proveditelné požadavky.

3.1.1 Kvalita softwaru

Kontrola kvality softwaru má tři hlavní cíle: prvním cílem je vytvoření organizačních standardů a rámců, které by měly vést k tvorbě kvalitního softwaru, dalším cílem je aplikace konkrétních procesů kvality a posledním je tvorba plánu kvality projektu, což znamená určení cílů kvality projektu a definice toho, které standardy a procesy se budou používat.[13]

Kvalita nebo jakost neni u softwaru jednoduše definovatelný pojem. Pro srovnání ve výrobní oblasti není problém stanovit, zda daný výrobek má požadovanou úroveň kvality či ne. Pro určení kvality se zde využívá podrobná specifikace produktu a tzv. systém tolerancí, ten určuje kdy je daný výrobek ještě v mezích použitelnosti a kdy už ne. Pokud se daný výrobek nachází v těchto mezích, pak je vyhodnocen jako přijatelný. Na druhou stranu software je nehmotný digitální systém a pro stanovení určité úrovně kvality na něj nelze uplatnit princip tolerancí.[13]

Vývoj softwaru není mechanický proces, z čehož vyplývá, že použití standardizovaných procesů kvality a standardů kvality nemusí nutně vést ke kvalitnímu výsledku. Tvorba softwaru je totiž kreativní proces, který hodně závisí na znalostech a zkušenostech jednotlivých vývojářů, proto může standardizace procesu vývoje softwaru narušit kreativitu a tím snížit kvalitu softwaru.[13]

Abychom byli schopni objektivně zhodnotit kvalitu softwaru, musíme se zaměřit na to, zda software splňuje svou specifikaci. To znamená, že potřebujeme určit požadavky na softwarový systém. Požadavky na software jsou většinou výsledkem mnoha zainteresovaných osob ze strany zákazníků a je velmi těžké vyhovět všem. Ve výsledku se pak některým může zdát, že software je méně kvalitní jen proto, že nesplňuje to, co chtěli. Taktéž je důležité, aby zákazníci a vývojáři spolu vyřešili rozdílnou interpretaci stejných požadavků. Mezi další vlastnosti kvality patří i těžko

měřitelné vlastnosti jako jsou např. možnosti údržby softwaru nebo efektivita. Z výše uvedených vlastností vyplývá, že proces hodnocení kvality je subjektivní, protože tým kontroly kvality zde spoléhá hlavně na vlastní úsudek, zda už software vyhovuje pro svůj předpokládaný účel použití a má už dosaženou přijatelnou úroveň kvality.[13, 14]

Posouzení zda už má software dostatečnou úroveň kvality záleží na tom, jakých vlastností systému už bylo dosaženo, např. zdali byl software správně testován, jestli je použitelný, jestli má dostatečný výkon, zda byly dodržovány standardy programování a dokumentace při procesu vývoje, zdali je software spolehlivý a srozumitelný apod.[13]

Jedním z prvků posouzení kvality softwaru je ověření správné implementace všech požadavků pomocí testování. Tým kontroly kvality projde všechny provedené testy a záznamy a ověří, zda byly testy provedeny správně. Výsledky těchto testů umožňují týmu kontroly kvality určit, zda už systém plní požadované funkce nebo ne.[13]

Při posuzování kvality softwaru hraje důležitou roli uživatelská zkušenost - pokud reakce systému neodpovídají očekávanému chování systému, tak si uživatelé většinou dokáží najít jiný způsob, jak dosáhnout toho, co potřebují. Když je ale systém příliš pomalý či nespolehlivý, tak se už pro uživatele stává nepřijatelný.[13]

Je nemožné optimalizovat systém s ohledem na všechny požadavky - pokud například uživatelé požadovat robustnější systém, může se to odrazit na jeho výkonu, což nemusí být v souladu s očekáváním, proto se zavádí plán kvality, v němž jsou popsané nejdůležitější atributy systému. Pro případ, že nastanou nenadálé komplikace při vývoji, měly by být v plánu kvality stanoveny kritické atributy, které by systém měl splňovat, i když se tomu budou muset obětovat jiné funkce. V plánu kvality by též měla být uvedena definice hodnocení procesu kvality, což je dohodnutý způsob hodnocení, zda se produkt vyznačuje určitou kvalitou.[13]

Většina společností má vlastní standardy procesů, které používají při vývoji, ve větších společnostech je kontrola kvality softwaru zajišťována speciálním týmem zajišťujícím kontrolu softwaru. Tento tým řídí testování softwaru, odpovídá za kontrolu toho, že systémové testy pokrývají požadavky a že jsou o procesu testování uchovávány dostatečné záznamy.[13] Tým kontroly kvality by měl být nezávislý na vývojovém týmu, hlavním důvodem je, aby bylo zajištěno objektivní hodnocení kvality softwaru. Také by tento tým neměl patřit pod projektové manažery, kteří rozhodují o rozvrhu projektu a rozpočtu, protože kdyby nastaly neočekávané problémy při tvorbě softwaru, tak by se mohlo stát, že v rámci dodržení rozvrhu či rozpočtu projektoví manažeři zmírní požadavky na kvalitu softwaru. V menších podnicích je obvyklé, že tým kontroly kvality je zároveň i vývojovým týmem, je tedy zodpovědný nejen za vývoj softwaru, ale i za jeho kvalitu.[13, 15]

3.1.2 Softwarové standardy

Jsou důležitým prvkem pro zajištění kvality softwaru. Pro softwarový produkt se zvolí určité standardy, k nim se definují specifické procesy pro daný projekt, které umožní kontrolovat použití standardů a to zda byly dodržovány.[13]

Standardy dokumentace softwaru by měly obsahovat popisy všech důležitých postupů nebo procesů vývoje systému, jako jsou např. požadavky na systém nebo na kód a proces produkce softwarového systému apod.[13] Jejich charakteristickou vlastností je, že přesně určují organizaci různých typů dokumentů a též jejich formát. Důležitost standardů dokumentace spočívá v tom, že zajišťují, aby měly projektové dokumenty stejný vzhled a strukturu a usnadňují kontrolu toho, zda nebyl v dokumentaci vynechán důležitý materiál.[13]

Jsou tři hlavní důvody, proč je pro organizaci důležité používat softwarové standardy. Prvním důvodem je, že podnikové standardy obsahují znalosti o tvorbě softwaru, které jsou založené na zkušenostech s různými postupy tvorby softwaru a pro další použití se vybírají ty, které se ukázaly jako nejlepší či nejvhodnější pro řešení daných problémů. Druhým argumentem je, že standardy poskytují určitou oblast pro definici toho, co v určité situaci může být považováno za kvalitní, díky nim je pak jednodušší určit, zda už produkt dosáhl určité úrovně požadovaných vlastností nebo ještě ne. Třetím důvodem je zajištění souvislosti vývoje, využívá se toho, když práci jedné osoby převezme jiná osoba.[13]

Standardy produktů by se měly navrhovat s ohledem na to, aby bylo možné je aplikovat a kontrolovat způsobem nenáročným na čas a úsilí tomu věnované. Při kontrole kvality softwaru se používají dva typy standardů - prvním jsou standardy produktu, které se týkají vyvíjeného softwarového produktu (např. standardy dokumentů a dokumentace, standardy kódování, atd.) a druhým jsou standardy procesů, jež definují procesy, kterými je nutno se řídit při tvorbě softwaru, jako jsou např. optimální vývojářské postupy, definice procesů popisu, návrhu, validace atd.[13]

Kromě podnikových standardů kvality pro vývoj softwaru lze využít i mezinárodních standardů pro kvalitu ISO 9001, vztahujících se na obecné principy kvality, popisy procesů kvality a stanovení organizačních standardů a procedur.[13] Standardům ISO 9001 se podrobněji věnuji v kapitole 2.

3.1.3 Revize a inspekce kvality

Text následujících odstavců je převzat z [13].
Jsou činnosti sloužící k zajištění kvality, jejich hlavním úkolem je kontrolovat kvalitu výstupů projektu, zkoumají software, dokumentaci softwaru a záznamy o procesech, předmětem jejich hledání jsou chyby v softwaru nebo kódu a kontrola dodržování standardů.[13]

Revize

Úkolem revizí je vyhledávání chyb v celém procesu vývoje softwaru, neslouží jako hodnocení výkonnosti jednotlivých členů týmu, ale hledají potenciální problémy v procesu vývoje softwaru na úrovni dokumentace procesů, konzistenci a celistvosti revidovaných dokumentů a kódu, dodržování standardů kvality, specifikace softwaru, plánovaných testů, modelů procesu atd.[13]

Výsledky zjištěné revizním týmem se zaznamenávají a na základě vyhodnocení projektu revizním týmem mohou projektoví manažeři plánovat a rozdělovat prostředky do potřebných oblastí vývoje. Tento postup revizí se uplatňuje hlavně u neagilních metod programování.[13]

Při agilním vývoji softwaru má revizní proces trochu jinou podobu. Agilní vývoj softwaru se zaměřuje hlavně na rychlý vývoj kódu, používá metodu inkrementálního vývoje (inkrement - přírůstek, část). Při tomto vývoji společnost vytvoří určitý základ softwarového produktu dle daných požadavků, každých pár týdnů pak uvolní nové části stávajícího systému zákazníkům k vyzkoušení. Tímto způsobem si vývojáři zajistí rychlou zpětnou vazbu a dokáží přiměřenou rychlostí přizpůsobovat software měnícím se požadavkům.[13]

Agilní metody nepoužívají standardy či standardizované postupy při vývoji softwaru, revizní kontroly proto u nich mají neformální charakter, např. při uvolnění nové části softwaru proběhne revizní schůzka, při které se prodiskutují otázky a problémy související s kvalitou. U agilních metod je vhodné zkombinovat kontroly kvality s plánovaným vývojem, protože kvůli neformálnosti revizního procesu by kontroly mohly způsobit zpomalení vývoje softwaru.[13]

Inspekce

Jsou neformální kontroly softwaru prováděné členy vývojového týmu, kteří spolupracují při hledání chyb. Inspekce slouží jako doplnění testování, které nevyžaduje spouštění programu. Při inspekci programu jednotliví členové vývojového týmu procházejí po řádcích zdrojový kód a hledají různé vady a problémy. Mezi nejběžnější chyby patří logické chyby, chybné podmínky, vynechané funkce v kódu apod.[13]

Hlavní výhody inspekcí softwaru oproti testování softwaru:

- Inspekce se nezabývají interakcemi mezi chybami.
- Inspekcemi lze kontrolovat i nedokončené verze systému, pro testování nedokončených verzí softwaru by bylo zapotřebí vyvinout specializovanou testovací strukturu, což by bylo časově náročné a zvyšovalo by to náklady na vývoj.
- Inspekce kromě hledání vad v softwaru, též kontrolují soulad se standardy, schopnost údržby, nebo přenositelnost.

Inspekce v softwaru vyhledávají vady, jako jsou nevhodné algoritmy, neefektivní místa či špatný styl programování. Toto jsou jedny z hlavních nedostatků, které by v budoucnu mohly zkomplikovat údržbu či aktualizace softwaru.

Při inspekcích se též kontrolují tzv. kontrolní seznamy, které obsahují sbírky chyb a vad, které se nejčastěji vyskytují v kódu daného programovacího jazyka. Pro každý programovací jazyk je tento seznam jiný, protože každý jazyk umožňuje různou úroveň kontroly v době kompilace.[13]

Revize a inspekce jsou účinným nástrojem na detekci a odstranění chyb, ale používá je jen velmi málo společností, hlavním důvodem je ekonomické hledisko, protože revize a inspekce představují dodatečné náklady v průběhu vývoje softwaru.[13]

3.1.4 Měření kvality softwaru

Text následujících odstavců je převzat z [13].
Používají se zde softwarové metriky, což jsou vlastnosti softwaru, které lze objektivně změřit, příkladem těchto vlastností jsou: velikost softwarového produktu v řádcích kódu, index mlhavosti (čitelnost psaného textu), počet oznámených vad v produktu nebo počet dní, které zaberou jednomu člověku na tvorbu jedné softwarové komponenty.[13]

Jsou dva základní druhy softwarových metrik: kontrolní a prediktivní. Kontrolní metriky se zabývají správou procesů a softwarovými procesy, používají se hlavně při vylepšování kvality stávajících procesů, prediktivní metriky jsou někdy nazývané též produktovými metrikami, jejich hodnoty se zjišťují analýzou kódu softwaru, např. průměrná délka identifikátorů v kódu.[13]

Měření softwarového systému se dají využít buď k přiřazení hodnoty atributům kvality systému, což znamená, že pomocí měření vlastností systémových komponent lze vyhodnotit obtížně měřitelné vlastnosti systému jako např. možnost údržby, nebo se tato měření dají využít k identifikaci komponent, které mají nedostatečnou kvalitu, což jsou komponenty které se svými vlastnostmi například odchylují od normy nebo mají velmi složitou strukturu, u níž se předpokládá, že budou obsahovat chyby.[13]

Některé vlastnosti nelze měřit přímo a souvisí s tím, jak software vnímají uživatelé a vývojáři, tyto vlastnosti se nazývají externí a k jejich změření potřebujeme interní vlastnosti, protože se vychází z předpokladu, že externí a interní vlastnosti spolu souvisí. Abychom mohli změřením interní vlastnosti předpovědět externí vlastnosti softwarového systému, musíme splnit tři podmínky: první podmínkou je nutnost přesného měření interní vlastnosti, druhou podmínkou je existence vztahu mezi vlastností, kterou lze změřit a externí vlastností kvality, která je v oblasti našeho zájmu a třetí podmínkou je vyjádření tohoto vztahu mezi dvěma vlastnostmi pomocí vzorce či modelu.[13]

Přínos systematického měření softwaru by měl spočívat v tom, že společnosti získaná data využijí ke zlepšení kvality svých produktů, jenže málokterá společnost tyto metriky zavádí. Hlavními důvody proč je složité zavést tyto metriky, jsou:

- při zavedení organizačního programu metrik je problém s kvantifikováním návratnosti investic do tohoto systému,
- neexistují standardizované procesy měření, analýzy nebo standardy softwarových metrik,
- spousta společností nemá softwarové standardy ani definované standardizované procesy,
- výzkum metrik se soustřeďuje hlavně na metriky kódu a procesy plánovaného vývoje, což může být problém, protože spousta společností v současnosti stále více využívá agilních metod programování, což nemusí být kompatibilní s měřením metrik.[13]

Produktové metriky

Slouží k měření interních vlastností systému, jako je např. velikost programu v počtu řádků, jejich problémem je určení jejich vztahu k externím vlastnostem, jako je možnost údržby a nebo srozumitelnost.[13]

Máme dvě třídy produktových metrik: dynamické a statické. Dynamické metriky se získávají měřením spuštěného programu, lze je získávat testováním nebo nasazením systému do praxe, tyto metriky pomáhají hodnotit spolehlivost a efektivitu programu a jako příklad lze uvést počet nahlášených chyb za jednotku času.[13] Statické metriky se získávají měřením určitých zástupců systému, jako je dokumentace, návrh či část programu, mají svůj podíl na hodnocení srozumitelnosti, složitosti nebo možnosti údržby softwaru, jako příklad statických metrik lze použít délku kódu, index mlhavosti.[13]

Analýza softwarových komponent

Umožňuje analýzu jednotlivých systémových komponent pomocí metrik, jejich naměřená hodnota se porovnává s dalšími hodnotami komponent případně i s dříve naměřenými daty a pokud nalezneme anomálie v měření, které se velmi odlišují od normy, pak mohou svědčit o problémech s danými komponentami.[13]

Proces měření komponent má pět fází:

- Zvolení provedených měření - formulace otázek, na které by mělo měření odpovídat a stanovit měření potřebná k zjištění příslušných odpovědí, měření, která nesouvisí přímo s těmito otázkami, není nutno provádět.[13]
- Výběr komponent, které se budou hodnotit - záleží na tom, co je zapotřebí, někdy stačí vybrat reprezentativní vzorek komponent k měření, jehož hodnocení postačí k zjištění kvality celého systému, jindy může být užitečné upřednostnit klíčové komponenty systému, které se používají stále před komponentami, které se používají příležitostně.[13]

- Měření vlastností komponent - vybrané komponenty se změří a následně proběhne výpočet příslušných hodnot metrik.[13]
- Identifikace anomálních měření - vykonaná měření komponent porovnáme s předchozími hodnotami a mezi sebou, snažíme se mezi nimi najít hodnotové anomálie, podezřelé jsou nezvykle vysoké či nízké hodnoty každé jednotlivé metriky, tyto hodnoty pak mohou znamenat problémy s náležitými komponentami.[13]
- Analýza anomálních komponent - jednotlivé komponenty, u kterých vyšly anomální hodnoty některých metrik, dále analyzujeme, abychom zjistili, zda se mohou tyto hodnoty negativně podepsat na kvalitě jednotlivých komponent či ne. Například anomální hodnota metriky složitosti, může mít vysokou hodnotu, naznačující, že daná komponenta je náchylná ke vzniku chyby, ale tato hodnota metriky neurčuje, jestli je daná komponenta chybová a má sníženou kvalitu.[13]

Data získaná jednotlivými měřeními je vhodné shromažďovat a uchovávat, později mohou být velmi přínosná pro danou organizaci. Když má společnost už dostatečně velkou databázi měření, může pak porovnávat kvalitu softwaru mezi jednotlivými projekty a validovat vztahy mezi jednotlivými interními atributy komponent a charakteristikami kvality.[13]

3.1.5 Spolehlivost systému

Text následujících odstavců je převzat z [13].
Spolehlivost je v současnosti jedna z nejdůležitějších systémových vlastností, vyjadřuje míru důvěry uživatele v to, že systém bude pracovat dle jeho očekávání a též, že nastane, co nejméně systémových selhání. Je důležité rozlišit, která část systému nám selhala, nejčastějšími příčinami špatné funkce hardwaru jsou selhání komponent v důsledku špatného návrhu, výrobních chyb a nebo z důvodu konce jejich životnosti. U softwaru mohou nejfrekventovaněji za selhání chyby v implementaci, specifikaci nebo v návrhu. V současné době jsou nejběžnější příčinou selhání systému provozní selhání, jsou to neobvyklé reakce systému na to, jak jej uživatelé ovládají či používají.[13]

Spolehlivost systému má čtyři základní vlastnosti:

- Dostupnost systému - míra schopnosti systému být v libovolnou dobu funkční a schopný poskytovat uživatelům potřebné služby.[13]
- Spolehlivost systému - míra schopnosti systému správně poskytovat služby, dle očekávání uživatele.[13]
- Bezpečnost systému - posouzení toho, s jakou pravděpodobností způsobí systém škodu uživatelům nebo svému okolí.[13]
- Zabezpečení systému - míra schopnosti systému odolat úmyslným nebo náhodným vniknutím.[13]

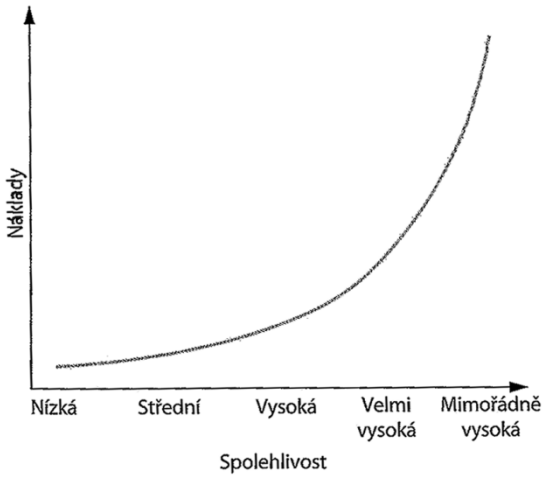
Mezi další systémové vlastnosti spolehlivosti patří též:

- Opravitelnost - je to vlastnost, která zajišťuje možnost diagnostikování problému, přístup k porouchané komponentě a možnost provedení změn vedoucích k opravě této komponenty v co nejkratším čase od vzniku poruchy, aby se minimalizovala doba narušení provozu způsobená tímto selháním.[13]
- Schopnost údržby - je to schopnost přizpůsobovat systém novým požadavkům tak, aby byl i nadále užitečný. Zároveň má nízkou pravděpodobnost zavedení nových chyb do systému v důsledku provedených změn.[13]
- Odolnost systému - je to schopnost systému pokračovat v poskytování služeb i v případě, že je systém vystaven útoku nebo je některá část systému vyřazena z provozu. Odolnost je možné zvýšit třemi způsoby: pomocí zvýšení odolnosti před útoky, rozpoznání útoku a zotavení systému ze škod, způsobených útokem.[13]
- Tolerance k chybám - reflektuje do jaké míry je systém navržen tak, aby neumožňoval chybný uživatelský vstup nebo jej toleroval, pokud dojde k chybě ze strany uživatele, měl by systém být schopen takovou chybu detekovat a buď ji opravit a nebo požádat uživatele, aby zadal hodnotu znovu.[13]

Pokud chceme vyvinout spolehlivý systém, tak měli bychom splňovat tyto požadavky: vyhnout se zavedení náhodných chyb do systému během specifikace a vývoje softwaru, navrhnout procesy verifikace a validace tak, aby odhalovaly zbytkové chyby, mít ochranné mechanismy, které tvoří prevenci před externími útoky, mít nasazený systém nakonfigurován spolu s podpůrným softwarem tak, aby v daném provozním prostředí fungoval správně.[13]

Aby bylo možné zajistit odolnost proti selhání systému, obsahují systémy redundantní (nadbytečný) kód, jehož funkce může být např. monitoring systému, předcházení selhání nebo zjišťování stavů, které mohou způsobit neočekávané rekce systému.[13]

Spolu se zvyšováním spolehlivosti systémů dochází k narůstání nákladů na vývoj, jak je vidět na obr. 2.



Obr. 2: Závislost nákladů na vývoj na požadované spolehlivosti.[13]

Z obr. 2 vyplývá, že pokud je software málo spolehlivý, pak náklady na zvýšení jeho spolehlivosti budou relativně malé při použití lepších metod softwarového inženýrství. Pokud už od počátku používáme optimálních postupů, pak náš systém má vyšší spolehlivost a další náklady na jeho možné zvýšení jeho kvalit jsou vyšší i když zlepšení nemusí být tak znatelná.[13]

3.1.6 Bezpečnost a zabezpečení systému

Text následujících odstavců je převzat z [13].

Bezpečnost

Je to systémová vlastnost, která reflektuje schopnost systému normálně nebo neobvykle fungovat, aniž by došlo ke škodě na životním prostředí či zranění osob, vyžaduje se hlavně u kritických systémů, což jsou to systémy, u nichž kdyby došlo k selhání, tak by to mohlo způsobit zranění osob, velké ekonomické ztráty, poškození majetku nebo životního prostředí. Příkladem bezpečnostně kritických systémů je například systém vesmírné navigace, systém převodu peněz, systém řízení procesů v chemických továrnách apod.[13]

Bezpečnostně kritický software má dvě hlavní odvětví:

- Primární bezpečnostně kritický software - to je software, který působí, jako řídící prvek systému do kterého je nasazen, chyba tohoto softwaru může vést k chybě hardwaru, která může mít za následek např. zranění osob nebo poškození životního prostředí.[13]
- Sekundárně bezpečnostně kritický software - je to software, který pokud selže, tak může způsobit zranění nepřímo, jako příklad lze uvést software na tvorbu návrhů konstrukcí staveb, pokud dojde k chybě softwaru, může to mít za následek vady navrhovaných objektů a následná zranění osob.[13]

Bezpečnost a spolehlivost spolu souvisejí, nemusí vždy ale znamenat, že spolehlivý software je i bezpečný nebo naopak. Existuje několik důvodů, proč spolehlivý software nemusí být bezpečný:

- Nikdy nemáme 100% jistotu, že software neobsahuje vady, proti kterým je odolný, některé vady se mohou objevit až po mnoha letech bezproblémového provozu.[13]
- Specifikace systému nemusí být úplná, protože někdy nepopisuje očekávané chování systému v kritických situacích.[13]
- Poruchy hardwaru mohou způsobit poruchy softwaru, z důvodu jiného pracovního prostředí, než na jaké je software nakonfigurován.[13]
- Může se stát, že uživatelé budou vkládat do systému nesprávné vstupy, které mohou vést k nesprávným reakcím systému.[13]

Existují tři způsoby, jak udržet bezpečnost:

- Prevence nebezpečí - systém je navržen tak, aby bylo možné předejít nebezpečným situacím.[13]
- Rozpoznání a odstranění nebezpečí - kdy je systém navržen tak, aby rozpoznal možná nebezpečí a odstranil je dřív, než dojde k nehodě.[13]
- Omezení škod - systém obsahuje ochranné a opravné funkce, které zajišťují minimalizaci škod, které jsou následkem nehody. Nehoda je událost, která nastane tehdy, když selže několik prvků systému současně.[13]

Zabezpečení

Je to schopnost systému chránit se před externími útoky, které mohou být náhodné či úmyslné, selhání zabezpečení může vést ke ztrátě dat, porušení systému či jeho dat nebo k úniku dat neoprávněným osobám.[13]

Existuje spousta systémů, kde zabezpečení je nejdůležitějším prvkem spolehlivosti systému. Mezi tyto systémy patří například vojenské systémy, systémy elektronického obchodování nebo systémy, které zprostředkovávají výměnu a zpracování důvěrných informací.[13]

S příchodem internetu začalo být zabezpečení systémů aktuálním tématem. Bez připojení na internet se zabezpečení počítače týkalo většinou jenom kontroly toho, zda data nezneužívají oprávnění uživatelé systému. Ale s připojením systému do sítě vyvstanuly na povrch další možnosti, jak se dostat k datům, jako jsou například vzdálený přístup, útoky na systém pomocí různých virů či trojských koní, neoprávněné přístupy k systému a úprava jeho dat apod.[13]

Pro každý systém připojený do sítě se vyskytují tři typy hrozeb:

- Hrozby důvěrnosti systému a jeho dat - tyto hrozby mohou směřovat k neoprávněnému přístupu k citlivým údajům, osobám nebo programům.[13]

- Hrozby vůči integritě systému a jeho dat - vedou k poškození nebo zničení softwaru nebo jeho dat.[13]
- Hrozby dostupnosti systému a jeho dat - vedou k omezení přístupu oprávněných uživatelů k softwarovému systému nebo k jeho datům.[13]

Výše uvedené hrozby se málokdy vyskytují samostatně, většinou potká systém nějaká kombinace hrozeb, například útok způsobí nedostupnost systému, která zabrání možnosti aktualizovat informace, tím dojde k narušení integrity systému a je nutné systém odstavit, čímž se omezuje jeho dostupnost.[13]

Velká část zranitelností v sociotechnických systémech je způsobena více lidským selháním než technickými problémy. Lidé si často vybírají lehce uhádnutelná hesla a nebo pokud si je zapisují, tak je ukládají na místech, kde jsou snadno dostupná, správci systému nesprávně nastavují řízení přístupu, uživatelé nepoužívají ochranné programy apod.[13]

Ke kontrolám zabezpečení systémů patří:

- Prevence zranitelností - zajišťují, že útoky na systém nebudou úspěšné, toho se dá docílit návrhem takového systému, ve kterém nenastávají problémy se zabezpečením, např. vojenské systémy obsahující velmi citlivá data nejsou v rámci zabezpečení připojeny k internetu. Další možností předejítí zranitelnosti je šifrování, to znesnadní útočníkovi přečíst neoprávněně získaná data. Prolomení šifrování je možné, ale je to velmi finančně a časově náročné.[13]
- Rozpoznání a zneškodnění útoků - do systému se zahrnou takové funkce, které umožní kontrolovat chování systému a detekovat neobvyklé vzorce chování. V případě zjištění neobvyklého chování systému, lze například preventivně vypnout určité jeho části nebo omezit přístup k systému jen na některé uživatele.[13]
- Omezení expozice a obnovení systému - jsou to kontroly, které pomáhají systému k zotavení po napadení, patří sem například zálohování.[13]

V současných systémech je důležité, aby dosahovaly alespoň přijatelné úrovně zabezpečení, protože bez toho nelze mít jistotu, že systém bude bezpečný a dostupný, např. při poškození systému útokem. Neméně důležitá je též spolehlivost systému, protože pokud systém selže, tak je velmi těžké zajistit bezpečnost a zabezpečení.[13]

3.1.7 Specifikace spolehlivosti a zabezpečení systému

Text následujících odstavců je převzat z [13].

Patří sem funkční požadavky, které popisují ochranné funkce systému před selháními a externími útoky a funkce obnovení a kontroly, pak zde náleží též mimofunkční požadavky, které popisují požadovanou bezpečnost a spolehlivost systému.[13]

Základem pro tvorbu funkčních požadavků jsou obvykle podnikové a doménové principy, předpisy nebo zásady. Mají podobu negativních požadavků na vlastnosti

systému, neboli popisují, jaké chování systému je neakceptovatelné. Tyto požadavky není možné implementovat přímo, ale je potřebné je rozebrat na jednotlivé funkční požadavky.[13]

Specifikace požadavků řízená riziky

Text následujících odstavců je převzat z [13].

Vychází ze znalosti konkrétního prostředí a rizik, která mohou ohrožovat systém, patří sem popis nebezpečných událostí, pravděpodobnost s jakou mohou tyto události nastat a pravděpodobnost vzniku škod a jejich rozsah. Když jsou objasněny možné příčiny systém ohrožujících situací, pak lze stanovit konkrétní požadavky na spolehlivost a zabezpečení.[13]

Specifikace požadavků řízená riziky se používá hlavně u systémů, kde zabezpečení a bezpečnost hrají kritickou úlohu. Soustřeďuje se na události s velmi pravděpodobným výskytem nebo na ty, které mohou způsobit velké škody v systému, jako jsou nebezpečné stavy systému vedoucí k nehodám či externí a interní útoky na systém.[13]

Proces specifikace řízené riziky má několik fází:

- Identifikace rizik - rozpoznání možných rizik systému, závisí na prostředí a interakcích mezi systémem a neobvyklými podmínkami prostředí.
- Analýza a hodnocení rizik - rizika se posuzují jednotlivě, eventuální závažná rizika, jsou podrobena další analýze, některá rizika jsou v této fázi vyloučena z důvodu nízké pravděpodobnosti výskytu.
- Rozložení rizik - zkoumání jednotlivých rizik, zjišťování kořenových příčin, což jsou příčiny případného selhání systému.
- Redukce rizik - návrh postupů a metod, pomocí kterých lze zjištěná rizika snížit či odstranit.[13]

Rizika lze rozdělit na různé typy dle druhu rizika:

- Rizika pocházející z prostředí systému - pomocí předběžné analýzy systému se vyvinou počáteční požadavky na spolehlivost a zabezpečení systému
- Analýza rizik životního cyklu - odehrává se během vývoje systému.
- Analýza provozních rizik - zabývá se rizikami možných chyb operátorů a uživatelského rozhraní systému.[13]

3.1.8 Specifikace bezpečnosti

Text následujících odstavců je převzat z [13].

Cílem bezpečnostně kritických systémů je určit požadavky, které sníží pravděpodobnost, že systém selže. Jsou to prvotní požadavky na ochranu systému a

nevztahují se k normálnímu provozu. Při navrhování těchto požadavků je důležité nalézt správnou rovnováhu mezi funkčností a bezpečností systému. Proces specifikace bezpečnosti využívá k analýze rizik proces požadavků řízených riziky.

Identifikace nebezpečí

Cílem je rozpoznat různá bezpečnostní rizika, která mohou být různých druhů, jako například elektrická, biologická, chemická, fyzická nebo i radiační. Každý druh nebezpečí je posuzován nejprve samostatně a poté se posuzují různé kombinace nebezpečí a hodnotí se, které mohou být více či méně nebezpečné.

Hodnocení nebezpečí

Soustřeďuje se na určení pravděpodobnosti, s jakou se nebezpečí projeví a na následky nehody vztahující se k tomuto nebezpečí. Hodnocení rizik má tři kategorie, jejich výsledkem je posouzení přijatelnosti:

- Netolerovatelná rizika - jsou to rizika, která mohou ohrožovat lidský život. Systémy se navrhují tak, aby se tato rizika nemohla projevit nebo aby byla rozpoznána dříve, než dojde k nehodě, náklady na návrh těchto systémů jsou velmi vysoké.
- Rizika ALARP (as low as reasonably practical, překlad: riziko je tak nízké, jak je rozumně praktické) - tyto rizika nemají až tak závažné následky nebo se vyskytují s velmi malou pravděpodobností. ALARP rizika jsou připouštěna pouze tehdy, pokud je zmenšení rizika, které představují finančně náročné nebo nepraktické.
- Přijatelná rizika - jsou to taková rizika, jejichž vlivem vzniklé nehody způsobí jen malou škodu. Při návrhu systému je snaha co nejvíce minimalizovat tyto rizika, za předpokladu, že nedojde k dodatečnému navýšení nákladů nebo třeba prodloužení dodací lhůty systému.

Hranice mezi jednotlivými typy rizik hodně závisí na společenských faktorech. S vývojem společnosti dochází ke stále menší toleranci rizik, veřejné mínění může být důležitým ukazatelem toho, jestli je vhodné investovat další náklady na ochranu proti systémovým nehodám, protože i když může být pro firmu výhodnější tolerovat nějakou malou nehodu a pak uhradit následky vzniklé v důsledku této nehody, tak tento postup nemusí být vždy přijatelný z pohledu společnosti.[13]

Analýza nebezpečí

Je to proces zjišťování kořenových příčin nebezpečných stavů systému, cílem je určit jednotlivé události či jejich kombinace, které mohou vést k selhání. Nejpoužívanějším postupem k zjišťování nebezpečných stavů je analýza stromů vad.

Analýza stromů vad začíná od rozpoznaných nebezpečí, kde se snažíme zjistit příčiny vzniku těchto nebezpečí, tyto příčiny umístíme do kořene stromu a následně zjišťujeme možné stavy systému vedoucí k těmto příčinám nebezpečí, pokud je to

zapotřebí zjišťujeme ještě další předchozí stavy ke stavům již zjištěným, dokud nedojdeme ke kořenovým příčinám rizik. Rizika, která mají jen jednu kořenovou příčinu, mají daleko vyšší pravděpodobnost výskytu, než rizika vzniklá kombinací několika kořenových příčin.[13]

Stromy vad se nepoužívají jen k rozpoznání chyb softwarových systémů, mohou se používat také k rozpoznávání potenciálních hardwarových chyb, kde mohou pomáhat určovat požadavky na software, které zlepší odhalování hardwarových chyb a možnosti jejich řešení. Jako hardwarové chyby lze uvažovat například chyby různých senzorů nebo časovačů.[13]

Redukce rizik

Ve chvíli, kdy známe potenciální rizika a jejich kořenové příčiny, lze vydedukovat konkrétní požadavky na bezpečnost, které poskytnou možnost kontroly rizik a možnost zajištění systému před vznikem nehod. Redukce rizik k tomu používá tři strategie:

- Prevence nebezpečí - systém se navrhne tak, aby nebezpečí nemohlo nastat.
- Rozpoznání a odstranění nebezpečí - nebezpečí je tímto systémem rozpoznáno a odstraněno dříve než způsobí škodu.
- Omezení škod - systém má snahu minimalizovat následky nehod.

V praxi se většinou používá kombinace těchto postupů, v bezpečnostně kritických systémech se nepřípustná nebezpečí řeší tím způsobem, že se minimalizuje pravděpodobnost jejich výskytu a do systému se přidá ochranný systém.[13]

3.1.9 Specifikace spolehlivosti

Text následujících odstavců je převzat z [13].

Spolehlivost je měřitelná systémová vlastnost, což znamená, že se dá stanovit požadovaná úroveň spolehlivosti systému a lze monitorovat systém v čase a kontrolovat, zda bylo dané spolehlivosti dosaženo či ne.

Máme dvě skupiny požadavků na spolehlivost:

- Funkční požadavky - popisují systémové funkce, které poskytují prevenci, rozpoznání či toleranci vad v systému a zabezpečují, že tyto vady nezpůsobí selhání systému.
- Mimofunkční požadavky - jsou to kvantitativní požadavky, stanovují počet selhání systému při normálním používání systému nebo čas, kdy systém není uživatelům k službám.[13]

Specifikaci spolehlivosti systému lze založit na specifikaci řízenou riziky, kdy nejprve určíme rizika možných systémových selhání, poté je zanalyzujeme, jak z hlediska zjištění kořenových příčin možných nebezpečí tak, z hlediska nákladů a následků možných systémových selhání a nakonec vytvoříme kvantitativní specifikace

spolehlivosti, které určí akceptovatelné pravděpodobnosti výskytu odlišných typů selhání.[13]

Metriky spolehlivosti

Text následujících odstavců je převzat z [13].

Spolehlivost je svým způsobem pravděpodobnost, že nastane systémové selhání v určitém provozním prostředí. Na určení spolehlivosti se používají dvě metriky spolehlivosti a jedna metrika dostupnosti:

- POFOD (probability of failure on demand) - definuje pravděpodobnost, že zadání požadavku na službu způsobí selhání systému.[13] Jeho hodnota se vyjadřuje jako např. 0,0001, což znamená, že nastane selhání s pravděpodobností 1/10 000. Metrika POFOD se používá tam, kde selhání na vyžádání, může vést k závažným systémovým problémům.
- ROCOF (rate of occurence of failures) - stanovuje pravděpodobný počet selhání za určitou časovou jednotku nebo za určitý počet spuštění systému. Opačná metrika k ROCOF je metrika MTTF (mean time to failure), občas se používá též jako metrika spolehlivosti a určuje průměrný počet časových jednotek mezi jednotlivými selháními.[13] Metriku ROCOF je vhodné využít tam, kde jsou požadavky na systém zadávány pravidelně.
- AVAIL (availability) - určuje pravděpodobnost, že systém bude funkční při zadání požadavku na službu.[13]

Abychom mohli vyhodnotit spolehlivost systému, musíme posbírat a zaznamenat data o jeho fungování. Tyto data obsahují:

- počet systémových selhání, zohledňujících množství požadavků na systémové služby, slouží k výpočtu hodnoty POFOD
- počet transakcí nebo čas mezi systémovými selháními, celkový uplynulý čas nebo celkový počet transakcí, hodnoty ROCOF, MTTF
- čas zprovoznění systému po výpadku nebo opravy systému, hodnota AVAIL.[13]

Mimofunkční požadavky na spolehlivost

Text následujících odstavců je převzat z [13].

Jsou to kvantitativní požadavky charakterizující spolehlivost a dostupnost systému. Se stále rostoucí poptávkou po službách s nepřetržitým provozem, se stále více klade důraz na používání kvantitativních specifikací spolehlivosti systémů. Lze je stanovit pomocí metrik spolehlivosti. Stanovení mimofunkčních požadavků na spolehlivost má několik výhod:

- Umožní zúčastněným osobám uvědomit si, jakou spolehlivost systému potřebují, skrze pochopení existence mnoha různých typů systémového selhání a vysokých nákladů na dosažení vysoké úrovně spolehlivosti.

- Jsou základem zhodnocení, kdy je možné přestat s testováním systému, z důvodu už dosažené požadované úrovně spolehlivosti.
- Lze je použít jako možnost ohodnocení odlišných taktik návrhů systému, jež mají zajistit lepší spolehlivost systému.
- Pokud musí být systém před uvedením do provozu schválen nějakou vyšší úřední entitou, pak tyto požadavky mohou posloužit jako důkaz, že bylo dosaženo určité úrovně kvality.[13]

Velkým problémem u specifikace spolehlivosti pomocí metrik spolehlivosti je to, že lze velmi snadno přehnat požadavky a způsobit tím enormní zvýšení nákladů na vývoj a validaci systému. Pokud máme například hodnotu metriky POFOD nastavenou na 0,001 (1 selhání na 1000 požadavků), tak se může stát, že některým zainteresovaným osobám to nebude připadat jako dostatečné a budou požadovat vyšší spolehlivost systému. Zde ovšem záleží na tom, jak často jsou služby požadovány, pokud je služba požadována zřídkakdy, tak 0,001 je opravdu vysoká spolehlivost. Pokud by byla služba volána kupříkladu 2 000 krát za měřenou jednotku času, tak záleží na okolnostech a zainteresovaných osobách, zda budou ochotni tolerovat možné 2 za daný časový úsek nebo ne.[13]

Existují postupy, s jejichž pomocí se dá vyhnout přehnaným specifikacím spolehlivosti, jako například specifikace požadavků na spolehlivost a dostupnost pro různé typy selhání, kde se snažíme, aby závažná selhání měla menší šanci výskytu, jak drobná selhání systému, nebo pro různé typy služeb systému, kde selhání kritických služeb systému budou mít nastavenou nižší pravděpodobnost výskytu než selhání služeb, které mají pouze místní účinky na systém. Dále je důležité se rozhodnout, jestli se vysoké úrovně spolehlivosti systému nedá dosáhnout i jinými způsoby než definováním výsoké úrovně spolehlivosti systémových služeb.[13]

Požadavky na dostupnost systému se trochu liší od požadavků na spolehlivost, protože spolehlivost je důležité zajistit po celou dobu existence systému, zatímco dostupnost systému bývá většinou důležitá hlavně v době předpokládaného používání, jako například u podnikových systémů, kde se požaduje dostupnost systému v rámci pracovní doby.

Funkční požadavky na spolehlivost

Tyto požadavky popisují požadavky, které definují funkce a omezení, které přispívají ke zvýšení spolehlivosti systému. Funkční požadavky na spolehlivost jsou třech typů:

- Kontrolní požadavky - slouží k rozpoznávání vstupů systému, aby bylo možné včas zachytit nesprávné vstupy, ještě před jejich zpracování systémem.
- Požadavky obnovení - pomáhají při obnovení systému po selhání, vztahují se k uchovávání dat a kopií systému, aby bylo možné po selhání obnovit systémové služby.

- Požadavky na redundanci - zajišťují, že selhání jedné komponenty nezpůsobí úplný výpadek všech služeb.[13]

3.1.10 Specifikace zabezpečení

Text následujících odstavců je převzat z [13].
Do určité míry souvisí s požadavky na bezpečnost, vyjadřují se negativně, jako projevy nepřijatelného chování systému. Existuje spousta důvodů, proč je větší problém určit požadavky na zabezpečení systému než na bezpečnost, u systémů, kde uvažujeme požadavky na zabezpečení, musíme brát v úvahu to, že:

- se někdo snaží úmyslně zaútočit na systém a má určité znalosti o slabých místech systému
- může být problém nalézt kořenovou příčinu selhání, protože se ji útočník snaží skrýt
- vypnout systém nebo přerušit poskytování služeb nemusí být řešení, protože jsou útoky, jejichž cílem je způsobit výpadek systému či omezení služeb, pokud tedy vypneme systém, tak je útok považován za úspěšný
- útočník testuje zabezpečení systému pomocí mnoha útoků, a jak se dozvídá nové věci z reakcí systému, tak své útoky přizpůsobuje.[13]

Ke specifikaci požadavků na zabezpečení je vhodné použít specifikaci požadavků řízenou riziky, kde provedeme předběžnou analýzu rizik, analýzu rizik během životního cyklu systému a analýzu provozních rizik.[13]

Proces hodnocení rizika pro požadavky na zabezpečení je následující:

- Identifikace aktiv - rozpoznání systémových aktiv, které mohou potřebovat ochranu, jako jsou konkrétní systémové funkce, systémová data nebo samotný systém.
- Hodnocení hodnoty aktiv - odhad hodnoty rozpoznaných aktiv.
- Hodnocení expozice - hodnocení potenciálních ztrát, jako jsou odcizení informací, náklady na obnovu systému a ztráta dobré pověsti společnosti.
- Identifikace hrozeb - rozpoznávají se hrozby u systémových aktiv.
- Hodnocení útoků - k analýze lze použít stromy útoků, jsou podobné stromům vad, umožňují rozložit hrozby na jednotlivé útoky a tím pomoci zjistit, jakým způsobem bude útok probíhat.
- Identifikace kontroly - kontroly jsou mechanismy, které umožňují chránit aktiva (např. šifrování dat).
- Hodnocení proveditelnosti - hodnotí se technické možnosti provedení kontrol a náklady na jejich návrh, protože je neekonomické navrhovat drahé kontroly pro ochranu systémových aktiv nízké hodnoty.
- Definice požadavků na zabezpečení - určují se dle znalostí o expozici, hrozbách a hodnocení kontrol systému.[13]

3.1.11 Formální specifikace

Text následujících odstavců je převzat z [13].
Formální specifikace je časově náročný a nákladný proces, s jehož pomocí lze softwarové požadavky převést do formálního jazyka, který umožňuje jednodušší kontrolu specifikací softwaru. K popisu specifikací se využívá matematický model. Při návrhu těchto specifikací se provádí důkladná analýza požadavků, aby bylo možné definovat případné problémy s požadavky, výsledkem by měl být jednoznačný popis žádaných systémových funkcí.

Výhody použití formální specifikace při vývojovém procesu:

- Spoustu chyb v požadavcích lze rozpoznat už v počátcích vývojového procesu, kdy oprava těchto chyb není tak nákladná, jak v pozdějších fázích vývoje.
- Jazyk formální specifikace má formálně stanovenou sémantiku, což umožňuje automatickou kontrolu, zjišťování nekonzistentnosti požadavků či chybějících prvků.
- Při použití určitých metod lze formální specifikaci přetvořit na program za pomoci sekvence transformací, které dodržují správnost, což garantuje, že konečný program bude vyhovovat své specifikaci.
- Snížení nákladů na testování, protože verifikace programu proběhla dle jeho specifikace.[13]

Nevýhody použití formálních specifikací:

- Vzájemná neporozumění jednotlivým specifikacím různými pracovníky vývoje softwaru. Například softwaroví inženýři, kteří rozumějí formální specifikaci mohou mít problém pochopit aplikační doménu a tím způsobem, si nemusí být jisti, zda je daná formální specifikace odrazem systémových požadavků či ne.
- Náklady na tvorbu formální specifikace lze poměrně dobře vyčíslit, nedá se ale předem zjistit, kolik nákladů se jejím využitím ušetří v celém procesu vývoje. Z tohoto důvodu nebývá až tak často využívána v praxi.
- Formální specifikaci lze obtížně použít na rozsáhlé systémy, proto se většinou používá jen na vývoj kritických částí systému.
- Nejsou kombinovatelné s agilními metodami vývoje.[13]

3.1.12 Vylepšování procesů

V současné době je stále důležitější nejen kvalita dodávaného systému, ale také čas, za který je dodán a náklady na jeho tvorbu. Zákazníci totiž vyžadují dobrou kvalitu produktu, za co nejmenší náklady a při co nejkratším čase dodání, což jsou požadavky, které lze jen málokdy splnit současně. Softwarové společnosti postupně začínají

přijímat procesy zlepšování kvality, ale aby je mohly úspěšně aplikovat, musí rozumět dosavadním stavům procesů a poté se je snaží přepracovat tak, aby došlo ke zlepšení kvality nebo úspoře času a nákladů na vývoj.[13]

Používají se dva způsoby zlepšování procesů:

- Proces založený na zralosti procesů, u něhož je hlavním cílem lepší kvalita produktů, předvídatelnost procesů a zavedení optimálních postupů vývoje do organizace,
- Agilní přístup, který se zaměřuje na iterativní vývoj, rychlé poskytování žádaných funkcí a rychlé reakce na změny trhu či požadavků zákazníků.[13]

Na kvalitu výsledného softwaru mají vliv čtyři hlavní faktory - použitá vývojová technologie, náklady, čas a rozvrh, kvalita procesů a kvalita lidí. Kvalita pracovníků je v mnoha případech mnohem důležitější než kvalita procesů, protože jedinci, kteří jsou už zkušení a znalí v dané oblasti, dokáží vyprodukovat software vysoké kvality, ať už použijí jakýkoliv proces. Na druhou stranu pokud jsou vývojáři málo schopní či zkušení, tak se může stát, že i když použijí procesy vysoké kvality, tak výsledný software moc kvalitní nebude.[13]

Dalšími frekventovanými problémy souvisejícími s kvalitou procesů kvality nebo jejich výsledkem, jsou:

- Programátoři podceňující důležitost kvality nejen procesu tvorby, ale i výsledného softwaru, většinou předpokládají, že jakost softwaru zajistí v dostatečné míře testování, na druhou stranu odborníci spolupracující s vývojovým týmem, často nerozumí problematice daného software,
- Softwarové společnosti se kvalitě většinou nevěnují, ať už z důvodu ekonomického nebo z důvodu často se měnících požadavků zákazníka a tím i častých změn parametrů kvality,
- Uživatelé softwaru mívají problém definovat, co požadují a jaké jsou jejich představy o kvalitním produktu, proto se také softwarové společnosti ani nesnaží investovat peníze do zvýšení kvality software, protože softwarové společnosti vyjde levněji zaplatit nějaké to malé odškodné a opravit chybu, jak investovat velké částky do zavedení kvalitních procesů tvorby, revizí a inspekcí softwaru a dalších možností zvýšení kvality softwaru.[15, 13]

Proces zlepšování procesů

Text následujících odstavců je převzat z [13].

Proces je určitá posloupnost úkonů vedoucích k vytvoření produktu. Téměř každá softwarová společnost má vlastní procesy a postupy na tvorbu svých produktů, které jsou závislé na požadavcích zákazníků, typu vytvářeného softwaru, zkušenostech a dovednostech vývojářů, trhu atd. Ke zlepšení procesů je důležité vzít v úvahu i kulturu prostředí daného podniku a ne jen přijmout konkrétní metody a nástroje, které pomohou zlepšit kvalitu výsledného procesu. Dále je důležité si rozmyslet, z kterých hledisek

chceme proces vylepšit, pokud chceme např. vylepšit některou vlastnost systému, tak je důležité se rozhodnout, které vlastnosti jsou velmi důležité a které už méně, protože aby vyvíjený systém byl efektivní a kvalitní, je důležité najít vhodný kompromis požadovaných vlastností.[13]

Proces vylepšování procesů má tři fáze, které se cyklicky opakují:

- Fáze měření procesu - změří se vlastnosti aktuálního produktu, které budou v budoucnu použité jako podklad pro vyhodnocení zlepšení,
- Fáze analýzy procesu - vyhodnocení aktuálního procesu a rozpoznání slabých míst systému, v této fázi se vytváří mapy procesů, což jsou modely, které popisují dané procesy,
- Fáze změny procesu - navrhují se změny procesů, které řeší rozpoznaná slabá místa systému, poté se změny aplikují na daný systém a celý cyklus vylepšování se zopakuje.[13]

Proces zlepšení se těžko vyhodnocuje bez dat, která by zlepšení prokázala, což je velkým problémem pro společnosti, které teprve začínají zlepšovat své procesy. Aby tyto společnosti byly schopné analyzovat svá data, musí nejprve začlenit do systému procesy shromažďování dat a měření vlastností softwarového produktu.[13]

Měření procesů

Text následujících odstavců je převzat z [13].

Slouží k získávání kvantitativních dat o systému a lze s jeho pomocí hodnotit efektivitu procesů. Je to způsob, který umožňuje vytvářet doklady o daném procesu a jeho změnách. Máme tři druhy metrik procesu:

- Metrika sledující čas potřebný k dokončení konkrétního procesu.
- Metrika sledující prostředky, kterých bude zapotřebí pro konkrétní proces.
- Metrika sledující počty výskytů konkrétních událostí.

První dvě metriky pomáhají zjistit, zda provedené změny zvýšily efektivitu procesu či nikoli. Příkladem těchto metrik je měření úsilí a času potřebného k přechodu z jednoho bodu procesu do druhého, z výsledku měření by pak mělo být patrné, zda změněný proces snížil náklady na čas a úsilí nebo ne. Třetí metrika sleduje výskyt různých událostí v souvislosti se změnou procesu, např. zvýšený výskyt určitého druhu vad.[13]

Abychom věděli, které informace budeme potřebovat pro další analýzu a zlepšení procesů, tak je důležité odpovědět na několik otázek, jako např. proč potřebujeme zavést zlepšování procesů, jaké informace potřebujeme sesbírat, abychom mohli vyhodnotit dané procesy a nebo která měření procesů musíme provést.[13]

K určení těchto otázek nám pomáhá model GQM (Goal - Question - Metric), který nám dává určité vodítko k určení toho, která data je důležité sbírat a doporučuje potřebnost analýzy dat z hlediska otázky, která nás zajímá. Model GQM má tři základní typy otázek:

- Cíl - čeho se snaží společnost dosáhnout, všeobecnější cíle jako např. vyšší bezpečnost produktu či kratší čas vývoje,
- Otázka - přesnější určení cílů, identifikace konkrétních otázek, jako např. otázky ohledně nutného počtu testů na zjištění určitého počtu vad,
- Metrika - měření, která je nutné sesbírat, abychom mohli odpovědět na otázky a vyhodnotit, zda zlepšení procesů pomohlo k dosažení cílů.[13]

Analýza procesů

Text následujících odstavců je převzat z [13].

Je studium procesů, jejíž hlavním úkolem je snaha o porozumění klíčovým vlastnostem procesu a porozumění toho, jak zainteresované osoby vytvářejí tyto procesy v praxi. Pracuje se v ní s modelem procesu, ve kterém jsou popsané aktivity procesu a vstupy a výstupy daných aktivit. Analýza procesů je propojena s měřením procesů, protože pokud nemáme naměřená data, tak nemáme co analyzovat. Analýza má několik dílčích cílů:

- porozumění aktivitám daného procesu a vztahům mezi nimi
- ujasnit si vztahy mezi provedenými měřeními a aktivitami procesu
- aplikovat daný zkoumaný proces na srovnatelný proces na jiném místě stejné organizace nebo na idealizovaný proces stejného druhu.[13]

Na analýzu procesů se používají dvě techniky:

- Analýza formou rozpravy a rozhovorů - kdy, jsou pracovníci dotazováni na reálný stav procesů. Někdy tato analýza dělá na pracovníky dojem hodnocení, ti pak místo pravdivých odpovědí uvádějí odpovědi, o kterých si myslí, že jsou žádoucí.
- Analýza formou studia pracovníků při práci může pomoci odhalit některé aspekty procesů, které nemusí být zřejmé z dotazníků či rozhovorů, používá se tam, kde potřebujeme dopodrobna porozumět jednotlivým částím procesu.[13]

Změna procesů

Text následujících odstavců je převzat z [13].

Zde dochází ke změně stávajícího procesu, po aplikaci změn se jejich efektivita vyhodnocuje za pomoci měření procesu. Změna procesu má pět fází:

- Rozpoznání zlepšení - z výsledků analýzy procesu, určuje způsob, jak zvládnout problémy s kvalitou. K řešení problémů stávajících procesů může navrhnout nové procesy nebo jejich struktury, nástroje a modely.
- Priority zlepšení - hodnocení změn procesů a určení priorit jejich aplikace.
- Zavedení změn procesu - nasazení nových metod, nástrojů či procesů do systému a kontrola jejich kompatibility se stávajícími prvky systému.
- Školení - pracovníci se zúčastní školení, kde porozumí navrženým změnám a způsobům jejich realizace. Protože mnohdy bez školení nedokáží pracovníci

správně aplikovat změněné procesy do praxe a pak nedochází ke zlepšení kvality dle očekávání.

- Ladění změn - slouží k drobným úpravám procesů v průběhu aplikace vylepšených procesů tak, aby výsledné procesy splňovaly očekávání.[13]

Není praktické nasazovat do praxe více změn najednou, protože je důležité analyzovat dopad každé jednotlivé změny na celý proces. Dále se vyskytujícím problémem může být odpor ke změnám, kdy se pracovníci obávají, že dané změny nebudou fungovat, že jejich aplikace bude zdržovat vývoj apod. V některých případech se může stát, že pracovníci záměrně narušují proces aplikace změn či interpretují data tak, aby poukázala na neefektivnost procesu. Problémem může být také trvalost změn procesu, protože některé procesy se po inovaci za nějakou dobu vrátí do původní podoby, protože už není zapotřebí výhod, které inovace přinášela.[13, 15]

3.1.13 CMMI jako model zlepšování procesů

Text následujících odstavců je převzat z [13].
CMMI (Capability Maturity Model Integration) slouží jako model zlepšování procesů, aplikovatelný v různých prostředích. Vytvořil jej institut SEI (Software Engineering Institute) v USA. CMMI má dvě verze fázovou a průběžnou. Fázová verze vyhodnocuje organizační procesy vývoje procesu a managementu a dle jejich zralosti jim přiřazuje hodnoty od 1 do 5.[13] Průběžná verze má jemnější stupnici hodnocení než fázová verze, hodnotí 22 oblastí procesů, její stupnice zralosti procesů může nabývat hodnot 0 až 5.

Základní složky modelu CMMI:

- 22 oblastí procesu spadajících do několika kategorií (inženýrství, správa procesů, řízení projektu, podpora), zabývají se možnostmi a zlepšováním softwarového procesu.
- Každá oblast procesu má v modelu CMMI své určité cíle, které popisují požadovaný stav v této oblasti.
- Každý cíl má definovaný optimální postup, který popisuje cestu k dosažení daného cíle. Postup k dosažení cílů nemusí být ten, který doporučuje model CMMI, ale i libovolný postup vyhovující dané organizaci.[13]

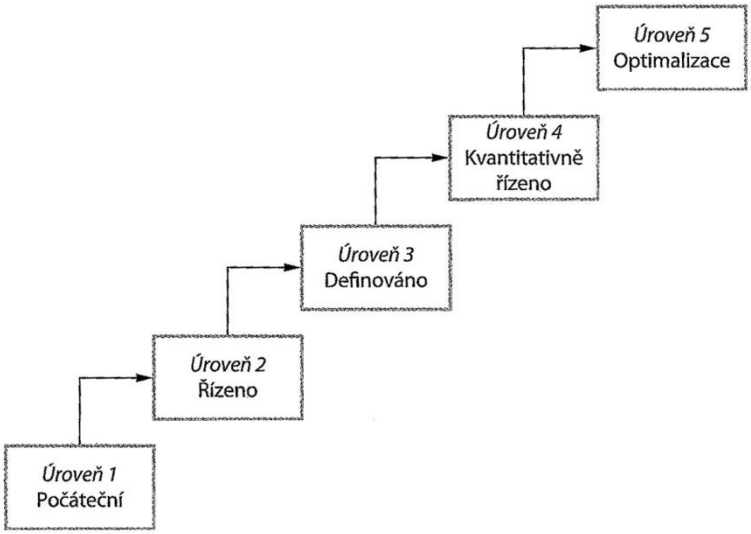
Úrovně zralosti při hodnocení procesu:

- Nedokončeno - není splněn minimálně jeden cíl z přidružených oblastí procesu, tato úroveň nemá obecné cíle.
- Provedeno - cíle přiřazené procesní oblasti jsou splněny a všechny dané procesy mají stanoven rozsah prováděných prací.
- Řízeno - cíle dané oblasti procesu jsou splněny, zavádějí se organizační zásady, které popisují, kdy je každý použit.[13]

- Definováno - nasazení procesů, organizační standardizace, potřeba kumulovat měření procesu a jeho aktiva k budoucímu použití ke zlepšení procesu.
- Kvantitativně řízeno - použití shromážděných dat měření procesu k řízení procesů.
- Optimalizace - nejvyšší úroveň, organizace musí za pomoci měření procesu a produktu dosáhnout zlepšení produktu, provádí se zde analýza trendů a procesy se přizpůsobují měnícím se obchodním potřebám.[13]

Fázový model CMMI

Má pět úrovní posuzování zralosti, každá úroveň má přidruženou oblast procesů a obecných cílů. Nižších úrovní zralosti lze dosáhnout použitím optimálních postupů, zatímco vyšší úrovně zralosti potřebují k optimálnímu stavu procesy měření a zlepšování.

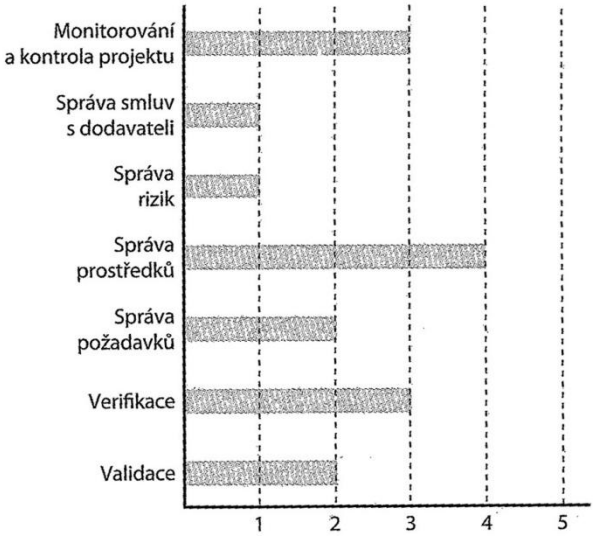


Obr. 3: Fázový model CMMI.[13]

Za nevýhodu fázového modelu lze považovat nutnost mít pro každou úroveň zralosti definované cíle a postupy implementovány ještě před postupem do další úrovně. Pro některé organizace může být ale vhodnější implementovat postupy z vyšší úrovně před postupy z nižší úrovně, tyto skutečnosti potom mohou velmi zkreslit hodnocení zralosti dané organizace.[13]

Průběžný model CMMI

Jednotlivé postupy jsou hodnoceny v rámci každé skupiny procesů, hodnocení zralosti zde poskytuje sadu hodnot, která umožňuje hodnotit zralost jednotlivých procesů či jejich skupin v rámci celé organizace.



Obr. 4: Průběžný model CMMI.[13]

Výhoda průběžného modelu oproti fázovému spočívám v tom, že společnost si může sama rozhodnout, které procesy jsou pro ni důležitější a potřebují zlepšit a zároveň mohou být společnosti lépe ohodnoceny modelem zlepšování CMMI.[13]

3.2 Testování software z hlediska prokazování jeho kvality

Text následujících odstavců je převzat z [13].
Testování je jedním z nejvíce používaných postupů k prokazování kvality softwaru. Cílem testování je odhalit vady systému před nasazením do praxe a ukázat budoucím uživatelům, že systém dělá jen to, co má. Program se při testování spouští s umělými daty, ve výsledcích testů se kontrolují chyby, anomálie nebo informace o mimofunkčních atributech kontrolovaného systému.[13]

Jsou dva hlavní cíle při testování softwaru: prvním cílem je ukázat uživateli nebo vývojáři, že software splňuje náležité požadavky, u obecného softwarového systému by měly být k dispozici testy všech systémových funkcí a jejich kombinací, které budou součástí vydání produktu a u menších zákaznických systémů postačí, když pro každý požadavek bude existovat alespoň jeden test.[13] Druhým cílem je zjistit situace, kdy se software nechová správně nebo žádoucím způsobem, nebo nesplňuje svou specifikaci, tyto možnosti mohou nastat v důsledku softwarových dat. Testování vad se soustřeďuje na odstranění nežádoucího systémového chování, k nimž patří: havárie systému, nesprávné výpočty, nežádoucí interakce s jinými systémy a poškození dat.[13]

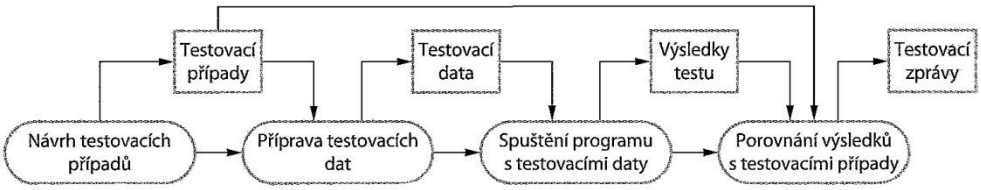
Testování je součástí procesů verifikace a validace, tyto procesy souvisejí s kontrolou, zda daný systém splňuje svou specifikaci a též, že systém splňuje funkce, které od něj uživatelé očekávají. Procesy verifikace a validace začínají se získáním požadavků od zákazníků a probíhají po celou dobu vývoje softwaru.[13]

Hlavní rozdíl mezi verifikací a validací je v tom, že verifikace je ověření, zda software vyhovuje schváleným funkčním i mimofunkčním požadavkům, zatím co validace slouží jen k zajištění toho, že software plní to, co od něj zákazník očekává. Z obou těchto procesů je důležitější validace, protože specifikace požadavků zákazníka nemusí vždy odrážet přání či potřeby systémových zákazníků a uživatelů.[13]

Cílem procesů validace a verifikace je přesvědčit uživatele, že softwarový systém vyhovuje svému účelu, to znamená, že by měl být dostatečně spolehlivý a důvěryhodný s ohledem na svoje předpokládané použití, úroveň důvěry závisí na očekávání uživatelů systému, účelu systému a marketingovém prostředí systému.[13]

Testování se v praxi provádí ručně, automaticky nebo kombinace obojího. Při ručním testování spouští tester program s testovacími daty a porovnává výsledky s očekávanými hodnotami, pokud zaznamená nějaké anomálie, tak je oznámí vývojovému týmu.[13] Automatické testování má testy zakódované v programu a jsou spouštěny vždy, když má být vyvíjený systém testován, tento postup je rychlejší než ruční testování, nevýhodou automatizovaných testů je, že kontrolují jen to, že program dělá to, co se od něj očekává, nelze těmito testy zkontrolovat či program nemá nějaké vedlejší účinky.[13]

Na obr. 5 máme model procesu testování, jež se používá nejčastěji v plánovaném vývoji. Testovací data se někdy generují automaticky, záleží na konkrétním zadání. Testovací případy podávají informaci o tom, co se testuje a specifikují se vstupy testu a očekávané výstupy testu (a výsledky), nelze je automaticky generovat, musí je vytvářet lidé, kteří rozumí tomu, jak se bude systém projevovat. Předpovězené výsledky testů se automaticky porovnají s výsledky proběhlých testů, takže není zapotřebí hledat chyby a anomálie ve výsledcích testů ručně.[13]



Obr. 5: Model procesu testování softwaru [13]

Softwarové systémy by měly projít třemi fázemi testování: vývojovým testováním, testováním vydání a uživatelským testováním.

- Vývojové testování zjišťuje chyby a vady v průběhu vývoje softwaru,
- Při testování vydání testuje testovací tým úplnou verzi softwarového systému před jejím uvolněním koncovému uživateli, cílem tohoto testování je zjistit, jestli softwarový systém naplňuje očekávání a požadavky budoucích uživatelů,
- Uživatelské testování probíhá, když uživatelé nebo potenciální uživatelé testují softwarový systém ve svém prostředí, zvláštním druhem uživatelského testování

je předávací testování, kdy zákazník testuje systém a rozhoduje se, jestli je systém už hotový a přijme ho nebo ho vrátí zpět do vývoje.[13]

3.2.1 Vývojové testování

Text následujících odstavců je převzat z [13].
Je to testování, které provádí vývojový tým systému, programátor, který software vyvinul nebo pár, který je tvořen programátorem a testerem, který vyvíjí testy a všeobecně pomáhá v procesu testování.[13]

Jeho hlavním úkolem je proces testování vad, který odhaluje chyby v softwarovém systému, obvykle se kombinuje s laděním, což je proces který lokalizuje problémy v kódu a pracovníci, kteří provádějí ladění, pak pomocí svých znalostí programování tyto problémy opraví.[13]

Vývojové testování může probíhat na třech úrovních:

- Testování jednotek - zaměřuje se na testování objektů nebo metod, kdy jsou testovány jednotlivé objektové třídy nebo programové jednotky.[13]
- Testování komponent - při tomto testování se spojuje několik samostatných jednotek do komponent a testuje se u nich hlavně jejich rozhraní.[13]
- Testování systému - je testování, kdy se spojují některé nebo všechny komponenty v celek, který je pak testován, předmětem testování jsou hlavně interakce mezi komponentami.[13]

Testování jednotek
Je to proces testování komponent programu, jako jsou jednotlivé objektové třídy či metody, jednodušším typem jsou pak funkce a metody, při testech se volají tyto činnosti pomocí různých vstupních parametrů.[13]

Testování jednotek je dobré automatizovat všude, kde je to možné, využívá se k tomu architektura automatizovaného testování (např. JUnit). Architektury testování vytvářejí generické testovací třídy, pomocí nichž lze vytvořit specifické testovací případy, umožňují taky spustit všechny implementované testy a oznamovat jejich úspěšné a neúspěšné výsledky.

Automatizovaný test sestává ze tří částí:

- Nastavení - kde se inicializuje systém pomocí testovacího případu, což jsou vstupy a očekávané výstupy.
- Volání - kdy se volá testovaný objekt nebo metoda.
- Potvrzení - výsledky volání se porovnávají s očekávanými výsledky. Když je potvrzení vyhodnoceno jako pravdivé, tak byl test úspěšný, v opačném případě je vyhodnocen jako neúspěšný.[13]

Hojně se využívá také testování cest, což je strategie testování, kdy se snažíme o průchod každou nezávislou cestou při provádění dané komponenty nebo programu.

Pokud projdeme každou nezávislou cestu, musí být všechny příkazy spuštěny alespoň jednou. Podmíněné příkazy se zde testují jako příkazy s hodnotou true / false.[13]

Volba testovacích případů jednotek

Je důležité vhodně zvolit testovací případy. Měly by v první řadě ukázat, že komponenta dělá to, co se od ní očekává nebo pokud komponenta obsahuje vady, tak by se měly při testování projevit.

Provádějí se většinou dva druhy testů, první z nich dokládá, že komponenta funguje při normálním běhu programu. Druhý typ testu je založen na zkušenostech testerů s tím, kde se obvykle vyskytují problémy. V tomto případě se pomocí abnormálních vstupů ověřuje, zda komponenta neselhává a zda jsou vstupy správně zpracovány. [13]

Testování softwarových komponent

Komponenty se skládají z několika spolu reagujících obvodů. K jejich funkcím lze přistupovat pomocí určeného rozhraní. V kompozitních komponentách se testuje zejména to, jestli se rozhraní komponent chová dle své specifikace.[13]

Různé programové komponenty mají různá rozhraní:

- Rozhraní parametrů - dochází zde k předávání dat nebo funkčních odkazů mezi komponentami,
- Rozhraní sdílené paměti - mezi komponentami je sdílen blok paměti, jeden z podsystémů ukládá data do paměti a druhý podsystém je odtud načítá. Toto rozhraní se často používá v integrovaných systémech, kde senzory vytvoří data, která se načtou a jiné komponenty je pak zpracují,
- Procedurální rozhraní - jedna komponenta do sebe zabalí sadu procedur, tyto procedury většinou volají jiné komponenty. Toto rozhraní se vyskytuje u opakovaně používaných komponent a objektů,
- Rozhraní pro předávání zpráv - komponenty spolu komunikují tím způsobem, že si předávají zprávy, např. se tímto mohou vzájemně požádat o spuštění služeb. Tato forma rozhraní se vyskytuje například u systémů typu klient/server.[13]

Chyby rozhraní patří mezi nejčastější typ chyb ve složitých systémech a je jich několik druhů. Nejběžnější je nesprávné použití rozhraní komponentami, kdy volající komponenta volá jinou komponentu a dopustí se chyby při použití jejího rozhraní, dalšími chybami jsou pak neporozumění rozhraní či chyby časování. Testování chyb rozhraní je obtížné, protože se objevují za neobvyklých podmínek, což může být například selhání jedné komponenty poté, co se jiná komponenta začala chovat nečekaným způsobem.[13]

Testování systému

Zaměřuje se na testování vztahů mezi objekty a komponentami, z nichž se systém skládá. Ověřuje kompatibilitu komponent, jejich interakce a to, jestli mezi rozhraními předávají správná data v patřičnou dobu.

Pokud systém vytvoříme integrací komponent, může se projevit jeho emergentní chování, což znamená, že některé systémové funkce lze pozorovat až při kombinaci komponent, např. při spojení autentizační komponenty s komponentou aktualizující informace, můžeme získat systémovou funkci, která umožní provést aktualizaci informací pouze oprávněným uživatelům.[13]

Při testování komponent nově přidaných do systému otestujeme tyto komponenty nejen samostatně, ale i s dříve testovanými komponentami, abychom zjistili, jestli se chovají v systému i samostatně dle očekávání.

Pro testování systému je vhodné použít testování založené na případech použití. Jednotlivé případy se implementují pomocí komponent a objektů v systému. Abychom mohli sledovat, jak se komponenty a objekty podílejí na interakci, je vhodné vytvořit sekvenční diagram k modelování implementace. Sekvenční diagram zobrazuje požadované vstupy a vznikající výstupy a tím pomáhá nabídnout konkrétní testovací případy. Každé testování systému se zakládá na určité podmnožině testovacích případů, protože z hlediska rozsáhlosti systémů je prakticky nemožné otestovat všechny možné sekvence provádění programu.[13]

3.2.2 Vývoj řízený testováním (test-driven development, TDD)

Text následujících odstavců je převzat z [13].

Je to způsob vývoje, kdy probíhá paralelně testování a vývoj kódu. Kód se vyvíjí inkrementálně spolu s testy těchto inkrementů. Další přírůstek programu je možné vytvořit až poté, co předchozí vytvořený kód projde testem. Vývoj řízený testováním se uplatňuje zejména u agilních metod programování, lze jej však použít i v plánovaných vývojových procesech. Dále je vhodné jej použít na vývoj nového softwaru, kde se funkce implementují pomocí otestovaných standardních knihoven nebo se přímo implementují v novém kódu.[13]

Výhody použití:

- pomáhá vývojářům lépe pochopit k čemu, která část kódu slouží
- zlepší pokrytí kódu tzn., že každý úsek kódu by měl mít alespoň jeden přidružený test,
- regresní testování - je využíváno ke kontrole, jestli nové přírůstky kódu nezanesly do systému nové chyby. Při regresním testování se nanovo spouští sady testů, které už dříve proběhly úspěšně, lze je spouštět kdykoli v průběhu vývoje,
- zjednodušené ladění - z výsledku neúspěšného testu je vidět, kde se vyskytuje problém, následně se zkontroluje a opraví kód programu,

- systémová dokumentace - testy slouží jako typ dokumentace, popisují, co by měl kód dělat.

Abychom snížili náklady na TDD testování je vhodné testy automatizovat, dále se toto testování používá např. pro validaci systému nebo aby se ukázalo, že systém nedělá nic, co by neměl.[13]

3.2.3 Testování vydání

Text následujících odstavců je převzat z [13].
Proces určitého vydání systému, které je určeno pro zákazníky a uživatele, případně pro jiný vývojový tým, který vyvíjí související software. Testování vydání provádí samostatný tým, který se nepodílí na vývoji softwaru.

Cílem testování vydání je ověřit, zda je systém připraven pro dodání zákazníkovi, tzn., zda splňuje požadavky zákazníka, neselhává při normálním používání, poskytuje určitý výkon a spolehlivost a obsahuje specifikované funkce. Testy se vyvozují v závislosti na specifikacích systému, používá se testování typu Black box (černá skříňka), kdy se chování systému určuje pouze ze sledování jeho vstupů a výstupů.[13]

Testování založené na požadavcích
Patří do kategorie validačního testování, snaží se ukázat, že systém správně implementoval své požadavky. Toto testování je přístup k návrhu testovacích případů takovým způsobem, že zvažujeme jednotlivé požadavky a vyvíjíme pro ně sady testů.

Požadavky je vhodné definovat tak, aby se pro každý požadavek dal vytvořit test, který ověří funkčnost tohoto požadavku. Pro každý požadavek je dobré vytvořit více než jeden test, protože je zapotřebí pokrýt všechny možné stavy při plnění požadavků. Při testování založeném na požadavcích je vhodné udržovat záznamy, které umožňují sledovat, které testy jsou spojeny s kterými testovanými požadavky.[13]

Testování scénářů
Je to typ testování vydání, kdy si stanovíme typické scénáře použití a k nim vyvineme poté testovací případy. Scénář popisuje způsob, jakým lze se systémem pracovat. Měl by být realistický a uživatelé by mu měli rozumět. Testování scénáře je dobré založit na věrohodné a propracované textové historii, která motivuje zainteresované osoby tím způsobem, že se stotožňují se scénářem a věří tomu, že je důležité, aby systém prošel testem. Z každého scénáře se testuje jen několik požadavků, kontroluje se jejich funkčnost a to, jestli kombinace nezpůsobuje problémy.[13]

Testování výkonu
Testuje se až už je dokončená systémová integrace. Testy se navrhují tak, aby prozkoumaly, zda systém dokáže bez selhání pracovat při určitém zatížení.

Pro testování výkonu si musíme sestavit operační profil, což je sada testů, které reflektují reálnou kombinaci úkolů, které bude systém zpracovávat. Pokud profil obsahuje více úloh v systému, tak do testů zahrnujeme hlavně kombinace nejvíce využívaných úloh, abychom dostali co nejpřesnější hodnoty reálného výkonu systému.

Testování výkonu se také nazývá stresové testování, důvodem je přístup k systému během testů. Průběh těchto testů je následující: nejprve se v systému spustí testování menšího počtu příkazů, za určitou časovou jednotku, než na kolik je systém zkonstruovaný. Pak se postupně zvyšuje množství příkazů, dokud se nedostaneme výrazně za maximální množství příkazů, ke kterému je systém zkonstruován a zkoumáme, za jakých okolností systém selhal a jakým způsobem. Při selhání systému je důležité, aby systém nezpůsobil poškození dat či ztrátu uživatelských služeb.[13]

Při vysoce zatíženém systému se také mohou projevit vady, které by se za normálních okolností neprojevily. Tímto způsobem mohou testující pracovníky upozornit na existenci možných kombinací faktorů, jež by mohly vést k selhání systému.

Stresové testování je hojně využívané zvláště u systémů, které se skládají ze sítě procesorů. Při vysokém zatížení je síť zaplavena velkou spoustou dat, které si procesory vyměňují. Tím jak vzájemně čekají na požadovaná data, se procesy celého systému zpomalují a dochází k velkému poklesu výkonu. Stresové testování analyzuje, kdy ke zpomalování začíná docházet, takže lze systém poté doplnit o funkce, které odmítnou přijímat nové úlohy, dokud nebudou zpracovány ty současné.[13]

3.2.4 Uživatelské testování

Text následujících odstavců je převzat z [13].
Je to fáze testování, kdy uživatelé či zákazníci přispívají svými názory do tvorby systému. Toto testování probíhá většinou mimo vývojové prostředí, protože je velmi těžké napodobit reálné prostředí nebo způsob, jakým budou uživatelé systém používat.

V praxi rozlišujeme tři druhy uživatelského testování: alfa testování, beta testování a předávací testování.

Alfa testování
Zahrnuje spolupráci budoucích uživatelů systému a vývojářů na testování systému v průběhu vývoje. Důležitost této spolupráce spočívá v tom, že uživatelé mohou poskytnout praktické informace o požadavcích na systém, mohou zodpovídat případné otázky vývojářů nebo rozpoznávat možné budoucí problémy se softwarem a tím vším pomoci navrhnout reálnější testy pro systém.[13]

Alfa testování se používá zejména u krabicových nebo podnikových systémů. Uživatelé těchto systémů se rádi podílejí na testování, protože se jim tím dostane informací o nových funkcích systému. Tímto testováním se omezuje riziko, že nové změny systému, by mohly špatným způsobem ovlivnit činnost produktu.[13]

Beta testování
Je testování, které provádějí uživatelé ve svém provozním prostředí. K tomuto testování se používá určitá verze systému, může být i nedokončená. Tyto testovací verze systému většinou vývojová společnost nabízí svým budoucím zákazníkům, aby si mohli vyzkoušet alespoň dosavadní vlastnosti systému ve svém prostředí a zároveň aby mohli poskytnout vývojářům zpětnou vazbu o funkčnosti systému a jiných jeho mimofunkčních vlastnostech v jiném jak vývojovém prostředí. Je spousta společností, které poskytují beta verze svých programů zdarma dostupné pro všechny uživatele, kteří mají zájem. Tyto společnosti tím kromě zpětné vazby o vlastnostech systému získávají potenciální zákazníky pro své stávající i nové produkty.[13]

Předávací testování
Je to souhrn aktivit, při kterých se zákazník rozhoduje, zda výsledný systém převezme od vývojářů a zaplatí nebo zda jej vrátí na dodatečné úpravy nebo se rozhodne systém vůbec nepřevzít.[13]

3.2.5 Verifikace a validace

Text následujících odstavců je převzat z [13].
Oba tyto procesy slouží k hledání různých chyb v softwaru, které by mohly vést ke zhoršené spolehlivosti, dostupnosti, bezpečnosti a zabezpečení systému. Procesy validace a verifikace by měly prokazovat zákazníkům, že se systém chová dle očekávání a že plní svou specifikaci. U kritických systémů jsou náklady na verifikaci a validaci vyšší než u jiných systémů, důvodem je, velmi striktní analýza a testování. Těmito způsoby se snažíme předejít systémovým selháním, jejichž nápravy by u kritických systémů byly velmi nákladné. Dále slouží k formálnímu doložení, že systém plní své požadavky na spolehlivost. V praxi může být důležité doložit, jakým způsobem byl systém verifikován a validován, podle výsledků těchto procesů může pak vyšší kontrolní orgán udělit certifikaci, že systém splňuje svou specifikaci a je bezpečný.[13]

Statická analýza
Je jedním z procesů verifikace a validace a umožňuje zaznamenat chyby v systému, ještě než bude vytvořena jeho spustitelná verze. K jejímu použití není zapotřebí spouštět kód, pracuje se zde s určitou reprezentací systému, jako je zdrojový kód či model návrhu a specifikace. Nejvíce používanými metodami statické analýzy jsou revize a inspekce programu, dále jsou to pak metody kontrolující chyby v použití UML jazyka, což může být například použití stejného názvu pro různé objekty. U kritických systémů se hojně využívá i dalších metod statické analýzy, jako jsou formální verifikace, kontrola modelu nebo automatizovaná analýza programu.[13]

- Formální verifikace - pomocí přesných matematických argumentů, dokládá, že systém vyhovuje své specifikaci.

- Kontrola modelu - kontrola formálního modelu systému, pomocí formálních tvrzení, která jsou v kódu uvedena jako komentáře. Ověřuje se, zda je program konzistentní s danými tvrzeními.
- Automatizovaná analýza programu - kontrola zdrojového kódu na vzory a modely, které bývají chybné. Užívá se zde kontrola charakteristických chyb, kontrola tvrzení a kontrola chyb definovaných uživateli systému.[13]

3.2.6 Testování spolehlivosti

Text následujících odstavců je převzat z [13].
Je to proces, který se snaží prokázat, že systém disponuje určitou spolehlivostí. Ke změření spolehlivosti lze použít metriky spolehlivosti, např. POFOD (pravděpodobnost, že systém selže na vyžádání) a ROCOF (četnost výskytu selhání).

Procesu testování spolehlivosti je někdy říká statistické testování a má čtyři fáze:

- Určení provozního profilu - provozní profil reflektuje použití systému v praxi, data k jeho určení získáme ze studia již existujících podobných systémů a analýzy toho, jak uživatelé vnímají tento systém a jeho spolehlivost. Pokud je systém, který vyvíjíme nového druhu, tak není prakticky možné vytvořit přesný provozní profil, protože nemůžeme vědět, jakým způsobem se bude systém používat.[13]
- Sestavení sady testovacích dat, která budou reflektovat provozní profil.
- Testování systému - výpočet počtu a typu selhání, která nastanou, zaevidujeme i časy těchto selhání.
- Posouzení statisticky významného počtu selhání a následný výpočet spolehlivosti a určení hodnot příslušných metrik spolehlivosti.

Při procesu testování spolehlivosti mohou nastat tyto potíže:

- Neurčitost provozního profilu - zkušenosti s použitím jiných, i když podobných systémů, nemusí reflektovat reálné použití vyvíjeného systému.
- Náklady na generování testovacích dat - pokud není naše testování plně automatizované, tak se může tvorba velkého množství dat, vyhovujících provoznímu profilu velmi prodražit.
- Statistická nejistota - při testování je snaha vygenerovat statisticky významné množství selhání, aby bylo možné prokázat vysokou spolehlivost systému. Pokud už vyvíjený systém spolehlivý je tzn., nastává málo selhání tak může být velký problém generovat nová testovací data.
- Identifikace selhání - pokud používáme formální specifikaci, tak není problém rozpoznat anomálie od této specifikace, pokud je ovšem specifikace napsána v přirozeném jazyce, může být problém zjistit, zda daný jev lze považovat za selhání nebo za normální chování systému.[13]

Statistické testování se často používá s tzv. injekcí vad, což je úmyslné vnesení chyb do systému a následná analýza vad systému a jeho selhání. Zkoumá se, zda je příčinou selhání chyba, která byla přidána. Toto testování odhalí určité procento chyb uměle zanesených do systému, které způsobí selhání a také by mělo pomoct odhalit určité procento skutečných chyb programu. Injekce vad je užitečný nástroj, ale není moc efektivní při hledání vad, které jsou způsobeny chybami v návrhu a požadavcích.[13]

3.2.7 Testování zabezpečení

Text následujících odstavců je převzat z [13].
Při testování zabezpečení se testuje hlavně rezistence systému vůči externím útokům. Často se na prozkoumání zabezpečení používá statická analýza, která může objevit chyby a možná slabá místa systému. Ke specifikaci zabezpečení se využívá negativních požadavků, které popisují nepřípustné chování systému. Testy se navrhují s ohledem na již existující typy útoků, nemohou ale pokrýt oblast nepředpokládaných či neznámých typů útoků. Tím, že útočníci vyvíjejí stále nové postupy a možnosti útoků, tak se může stát, že systém, který byl považován za bezpečný, bude napaden bez možnosti předejítí tomuto stavu.[13]

Na ověření zabezpečení se nejčastěji využívá těchto druhů testování:

- Testování založené na zkušenostech - přednostně se systém testuje na typy útoků, které testovací tým zná.
- Vytvoření speciálního testovacího týmu, který simuluje útoky na systém, snaží se narušit zabezpečení a hledat další nové způsoby narušení systému.
- Testování založené na nástrojích - k tomu se používá například statická analýza nebo analýza různých forem zajišťování bezpečnosti, jako je např. kontrola hesel (délka hesla, často používaná slova, stále se opakující znaky apod.).
- Formální verifikace - ověření systému pomocí formálních specifikací. U testování zabezpečení se tato možnost moc často nevyužívá.

V praxi se velmi často využívá proces testování zabezpečení založený na rizicích, kde se využívá analýza rizik zabezpečení daného systému, důvodem pro využití tohoto přístupu jsou omezené prostředky a čas vyhrazené na testování zabezpečení.[13]

3.2.8 Zajištění bezpečnosti

Text následujících odstavců je převzat z [13].
K zajištění bezpečnosti se v průběhu celého vývoje softwaru nejčastěji využívá analýzy bezpečnostních rizik, kde rozpoznáváme nebezpečí, možnosti výskytu a pravděpodobnosti vzniku nehod. Kontrolu nalezených nebezpečí a jejich řešení může

zajišťovat určitý kód programu, někdy se využívá i tzv. bezpečnostních argumentů, které by měly zamezit vzniku nebezpečných situací a nehod. Mezi aktivity zajištění bezpečnosti patří:

- monitorování a protokolování nebezpečí
- bezpečnostní revize
- bezpečnostní certifikace.

Zajištění těchto procesů provádějí nejčastěji inženýři projektové bezpečnosti, kteří nesou zodpovědnost za veškeré bezpečnostní hlediska systému. Jejich prací je zaručit, že nenastane systémové selhání v důsledku špatné bezpečnosti a též musí být schopni prokázat, že byly provedeny procesy vedoucí k bezpečnosti systému.[13]

K zajištění bezpečnosti je vhodné vytvořit dokument označovaný jako protokol nebezpečí, který v každé fázi vývoje systému definuje možná nebezpečí a způsoby, jak jim lze předejít.

Další využívané dokumenty jsou případy zajištění bezpečnosti a spolehlivosti. Mají formu strukturovaných dokumentů a charakterizují argumenty a důkazy, které zajišťují, že systém je bezpečný a že má požadovanou úroveň bezpečnosti a spolehlivosti. Je spousta druhů kritických systémů, kde je povinnost vytvářet tyto typy dokumentů. Na jejich tvorbu zde dohlíží určitý certifikační či regulační orgán, který nese odpovědnost za to, že hotový systém je bezpečný a spolehlivý, tak jak je to reálně možné. V těchto případech pracují regulátoři spolu s vývojovým týmem doporučují, co by mělo být složkou případu bezpečnosti, zkoumají procesy a postupy, aby bylo zajištěno jejich správné provedení a dokumentace.[13]

4 POŽADAVKY NA KVALITU SOFTWARE PODLE NORMY ČSN ISO/IEC 25 000

4.1 Struktura souboru norem ČSN ISO/IEC 25 000

Hlavním důvodem vedoucím k vytvoření souboru mezinárodních norem SQuaRE, bylo vytvořit logicky organizovaný celek, který bude pokrývat hlavní procesy kvality: specifikace požadavků na kvalitu a vyhodnocení kvality systémů a softwaru, pomocí procesů měření systémové a softwarové kvality.[16]

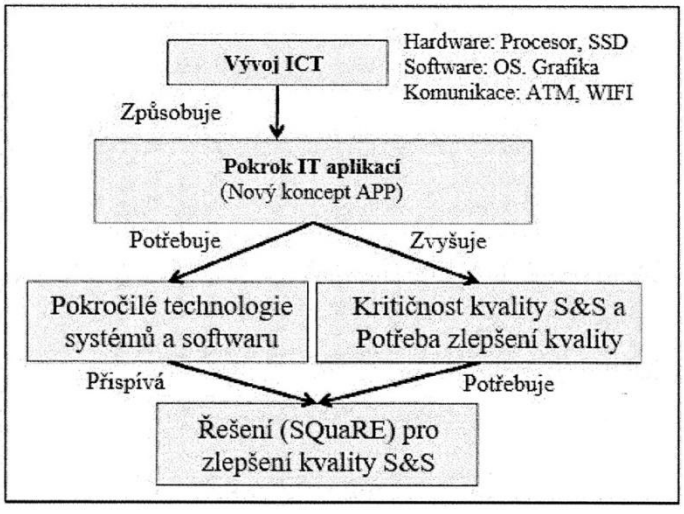
Jedním z podnětů k vytvoření norem SQuaRE, byly nesrovnalosti při používání norem ISO/IEC 9126 - Kvalita softwarových produktů a ISO/IEC 14598 - Vyhodnocování softwarových produktů.[16] Obě normy obsahují společné funkční, referenční a normativní základy, ale každá má svůj vlastní životní cyklus.[16]

Mezinárodní normy SQuaRE jsou nápomocné těm, kteří vyvíjí či nakupují softwarové a systémové produkty, jsou zaměřeny na specifikaci a vyhodnocování požadavků na kvalitu, stanovují kritéria pro specifikaci požadavků, měr a vyhodnocování produktů.[16]

Mezi hlavní výhody použití normy SQuaRE oproti ostatním normám kvality patří: soulad pokynů k měření a vyhodnocování kvality systémových a softwarových produktů, pokyny ke specifikaci požadavků na kvalitu a souhra s ISO/IEC 15 939, která má formu referenčního modelu měření kvality a je uvedena v ISO/IEC 25 020.[16]

Mezinárodní norma SQuaRE má několik částí: termíny a definice, referenční modely, obecné pokyny, pokyny k jednotlivým divizím a mezinárodní normy pro účely specifikace požadavků, plánování a řízení, měření a hodnocení.[16]

Nutností pro vytvoření řady norem SQuaRE byl hlavně rychlý vývoj informačních technologií, jak je vidět na obr. 6.



Obr. 6: Diagram vývoje informačních technologií a potřeby SQuaRE.[16]

S rostoucí složitostí a rozsáhlostí softwaru a systémů, se zvyšují nároky na kvalitu, odlišné systémy mají odlišné parametry kvality, které jsou závislé od prostředí, ve kterém se bude software používat nebo jaký bude účel jeho použití.[16]

Modely SQuaRE

Modely tvoří základ pro navigaci řadu norem SQuaRE, máme tři základní modely:

- Obecný referenční model - obsahuje pokyny pro navigaci řadou norem SQuaRE, závisí na rolích uživatele a jeho konkrétních informačních potřebách,
- Model životního cyklu kvality systémů a softwaru - nabízí pohledy na externí a interní kvalitu při použití během životních cyklů softwaru a systémů,
- Struktura modelu kvality - třídí charakteristiky kvality softwaru a systémů do podcharakteristik a atributů systému.[16]

SQuaRE model kvality se skládá ze tří částí, jimiž jsou: model kvality systémových a softwarových produktů, model kvality při použití a model pro kvalitu dat, tyto modely jsou podrobněji popsané v ISO/IEC 25 010.[16]

4.2 Charakteristika a obsah jednotlivých norem

Řada norem SQuaRE má pět hlavních oddílů a jeden oddíl navíc, který se zabývá možnými rozšířeními. Oddíly jsou následující: oddíl správy kvality, oddíl modelu Struktura souboru norem ČSN ISO/IEC 25 000 kvality, oddíl měření kvality, oddíl požadavků na kvalitu, oddíl hodnocení kvality, oddíl SQuaRE rozšíření.[16]

Oddíl Správy kvality

Jsou zde popsány všechny společné modely, termíny a definice odkazované ostatními normami z řady norem SQuaRE, dále tento oddíl obsahuje odkazy a návrhy, které se týkají aplikací správných norem na konkrétní aplikační případy, pokyny a požadavky pro funkci, která je zodpovědná za správu specifikace a hodnocení požadavků na výsledný produkt.[16]

Součástí tohoto oddílu jsou dokumenty: ČSN ISO/IEC 25 000 - Pokyny ke SQuaRE a ISO/IEC 25 001 - Plánování a správa.[16]

Oddíl Modelu kvality

Tento oddíl poskytuje detailní modely kvality pro systémové a softwarové produkty, kvalitu při použití, data a praktické pokyny k použití tohoto modelu.[16]

Zde jsou součástí oddílu dokumenty: ISO/IEC 25 010 - Model kvality a ISO/IEC 25 012 - Model kvality dat.[16]

Oddíl Měření kvality

Zahrnuje referenční model měření kvality systémových a softwarových produktů, definice měr kvality a praktické pokyny k jejich aplikaci.[16]

Patří sem dokumenty: ISO/IEC 25 020 - Referenční model a pokyn k měření, ISO/IEC 25 021 - Prvky měr kvality, ISO/IEC 25 022 - Měření kvality při použití, ISO/IEC 25 023 - Měření kvality systémových a softwarových produktů a ISO/IEC 25 024 - Měření kvality dat.[16]

Oddíl Požadavků na kvalitu

Obsahuje dokument ISO/IEC 25 030 - Požadavky na kvalitu, který poskytuje požadavky na proces, pokyny k procesu používanému ke stanovení požadavků na kvalitu a požadavky a doporučení ke kvalitě.[16]

Oddíl Hodnocení kvality

Poskytuje požadavky a doporučení k směrnicím na hodnocení kvality produktu, tyto hodnocení mohou provádět nezávislí hodnotitelé, nabyvatelé nebo samotní vývojáři systému, součástí tohoto oddílu jsou dokumenty: ISO/IEC 25 040 - Proces hodnocení, ISO/IEC 25 041 - Pokyny k hodnocení pro vývojáře, nabyvatele a nezávislé hodnotitele a ISO/IEC 25 045 - Moduly hodnocení obnovitelnosti.[16]

Oddíl Rozšíření

Patří sem tyto dokumenty: ISO/IEC 25 051 - Požadavky na kvalitu softwarového produktu připraveného k používání a instrukce pro testování, ISO/IEC 25 060 - Common industry format (CIF) pro použitelnost: Obecný rámec pro informace souvisící s použitelností, ISO/IEC 25 062 - Common industry format (CIF) pro použitelnost - zprávy z testování, ISO/IEC 25 063 - Common industry format (CIF) pro použitelnost: Popis kontextu použití, ISO/IEC 25 064 - Common industry format (CIF) pro použitelnost: Zpráva o potřebách uživatelů, ISO/IEC 25 065 - Common industry format (CIF) pro použitelnost: Specifikace uživatelských požadavků, ISO/IEC 25 066 - Common industry format (CIF) pro použitelnost: Zpráva o hodnocení použitelnosti.[16]

5 ZÁVĚR

Cílem této práce bylo charakterizovat požadavky na kvalitu, popsat problematická specifika softwaru, způsoby prokazování kvality a možnosti jejího zvyšování.

V úvodu jsem seznámila čtenáře s pojmem kvalita, proč je důležitá a jaká je její role ve vztahu k současným technologiím.

V druhé kapitole jsem se věnovala řadám norem ISO 9000 a ISO 9001, možnostem jejich využití při posuzování kvality produktu a evropskému přístupu ke kvalitě produktů.

V první části třetí kapitoly jsem podrobně analyzovala požadavky na kvalitu softwaru, možnosti měření splnění těchto požadavků, možnosti kontrol kvality v průběhu tvorby softwaru a metody zvyšování kvality softwaru.

V druhé části třetí kapitoly jsem se pak zabývala testováním softwaru, různými metodami testování a jejich použití při prokazování kvality softwarového systému.

V poslední kapitole se věnuji souboru mezinárodních norem ČSN ISO/IEC 25 000, které poskytují zatím nejucelenější podklady pro zlepšování kvality softwaru. Jejich obsahem je vymezení pojmů vztahujících se k softwaru a kvalitě, analýza různých modelů kvality v závislosti na jejich použití a praktické rady, které umožní uživatelům nebo hodnotitelům identifikovat správný způsob vedoucí ke zvýšení kvality softwaru.[16]

Kvalita softwaru začíná být v posledních letech stále důležitější, díky probíhající modernizaci velké části průmyslových odvětví a potřebám, aby procesy byly neustále k dispozici a pokud možno komunikovaly vzdáleně, se dnešní systémy neobejdou bez velmi dobrých úrovní spolehlivosti, bezpečnosti a zabezpečení, což jsou základní parametry kvality systémů.

Tato práce mi pomohla uvědomit si složitost tvorby softwaru, pochopit důležitost tvorby kvalitních systémů a proč je komplikované zavést procesy kvality do procesu vývoje softwaru. Znalosti získané vypracováním této práce se budu snažit využít při dalším studiu a v praxi.

6 SEZNAM POUŽITÉ LITERATURY

[1] *Co se skrývá pod výrazy Industry 4.0 / Průmysl 4.0 ?* [online]. Praha 4 - Kateřinky: HW server s.r.o. Poslední aktualizace: 19. 3. 2016. [cit. 2018-11-20]. Dostupné z: https://automatizace.hw.cz/mimochodem/co-je-se-skryva-pod-vyrazy-industry-40-prumysl-40.html

[2] *Kyberfyzikální systémy* [online]. Poslední aktualizace: 22. 8. 2016. [cit. 2018-11-20]. Dostupné z: https://www.iot-portal.cz/2016/08/22/kyberfyzikalni-systemy/

[3] ISO 9000 [online]. Poslední aktualizace: 20. 2. 2012. [cit. 2018-12-08]. Dostupné z: http://wiki.knihovna.cz/index.php/ISO_9000

[4] ČSN EN ISO 9000 (010300). Systémy managementu kvality - Základní principy a slovník. ÚNMZ. 1. 4. 2016

[5] NENADÁL, Jaroslav a kol.: *Moderní management jakosti*. Praha, Management Press, s. r. o., 2008. Počet stran 377. ISBN 978-80-7261-186-7

[6] ČSN EN ISO 9001 (010321). Systémy managementu kvality - Požadavky. ÚNMZ. 1. 3. 2016

[7] *ISO 9001* [online]. [cit. 2018-12-08]. Dostupné z: http://www.iso.cz/iso-9001

[8] LACKO, Branislav. *Navrhování systémů řízení.* Interní materiály UAI FSI VUT.

[9] Pojetí kvality a procesní přístup [online]. VŠCHT, Technická 5, Praha 6. [cit. 2018-12-08]. Dostupné z: https://web.vscht.cz/kocourev/files/VK323442_1.pdf

[10] VANÍČEK, Jiří. *Měření a hodnocení jakosti informačních systémů*. Praha: Credit, 2000. Počet stran 211. ISBN 80-213-0667-X

[11] *Overview of the U.S. Standardization System* [online]. [cit. 2019-05-18]. Dostupné z: https://www.standardsportal.org/usa_en/standards_system.aspx

[12] *Capability Maturity Model Integration* [online]. San Francisco (California): Wikimedia Foundation. Poslední aktualizace 15. 04. 2019. [cit. 2019-05-17]. Dostupné z: https://en.wikipedia.org/wiki/Capability_Maturity_Model_Integration

[13] SOMMERVILLE, Ian. *Softwarové inženýrství.* Brno, Computer Press, 2013. Počet stran 680. ISBN 978-80-251-3826-7.

[14] Systém řízení jakosti QMS - IS MU [online]. Masarykova univerzita, Žerotínovo nám. 617/9, Brno. [cit. 2018-12-08]. Dostupné z: https://is.muni.cz/el/1441/jaro2017/UOPK_0013/um/Uvod_do_jakosti.pdf

[15] LACKO, Branislav. Systémový přístup k jakosti softwaru. *Sborník z jubilejního 30.ročníku konference Tvorba software 2004*. VŠB-TU Ostrava a Tanger Ostrava 2004, str. 129-138. ISBN 80-85988-96-8.

[16] ČSN ISO/IEC 25000 (369006). Systémové a softwarové inženýrství - Požadavky a hodnocení kvality systémů a softwaru (SQuaRE) - Pokyn ke SQuaRE. ÚNMZ. 1. 8. 2018

7 SEZNAM ZKRATEK, SYMBOLŮ, OBRÁZKŮ A TABULEK

Seznam zkratek

ALARP (as low as reasonably practical) - riziko je tak nízké, jak je rozumně praktické
POFOD (probability of failure on demand) - pravděpodobnost selhání systému na požádání
ROCOF (rate of occurence of failures) - frekvence výskytu selhání
MTTF (mean time to failure) - průměrná doba do selhání
AVAIL (availability) - dostupnost
GQM (Goal - Question - Metric) - cíl-otázka-metrika
CMMI (Capability Maturity Model Integration) - model zlepšování procesů
SEI (Software Engineering Institute) - Institut softwarového inženýrství
TDD (Test-driven development) - Vývoj řízený testováním
ISO (International Organization for Standardization) - mezinárodní organizace pro normalizaci
IEC (International Electrotechnical Commission) - mezinárodní elektrotechnická komise
SQuaRE (System and Software Quality Requirements and Evaluation) - Požadavky a hodoncení kvality softwaru a systému
CIF (Common industry format) - společný průmyslový formát

Seznam použitých obrázků

8 SEZNAM PŘÍLOH

V přiloženém CD je tento dokument v elektronické podobě.