Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Klasifikace chyb SW na základě analýz z úložiště

Plzeň 2019 Bc. František Koleňák

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 20. března 2019

Bc. František Koleňák

Abstract

The text of the abstract (in English). It contains the English translation of the thesis title and a short description of the thesis.

Abstrakt

Text abstraktu (česky). Obsahuje krátkou anotaci (cca 10 řádek) v češtině. Budete ji potřebovat i při vyplňování údajů o bakalářské práci ve STAGu. Český i anglický abstrakt by měly být na stejné stránce a měly by si obsahem co možná nejvíce odpovídat (samozřejmě není možný doslovný překlad!).

Obsah

1 Úvod

V souboru `literatura.bib` jsou uvedeny příklady, jak citovat knihu [3], článek v časopisu [1], webovou stránku [2].

2 Řízení projektů

Řízení projektu (někdy též projektové řízení) se zabývá řízením projektu, tedy časově ohraničené a ucelené sady činností a procesů, jejímž cílem je co nejefektivněji zavést, vytvořit nebo změnit něco konkrétního (definováno daným projektem).

Tradiční přístup řízení projektů je založen na důkladném naplánování na začátku projektu a řízení všech aktivit v průběhu projektu. Tento přístup (též známý jako vodopádový model) je nejvíce vhodný na projekty, které mají podobu cíle jasně danou a neměnnou (chodník, dům) a ve kterých je nutné dobře naplánovat všechny aktivity související s daným projektem. Tradiční přístup vyžaduje kvalitně popsaný cíl, výstupy a plán projektu.

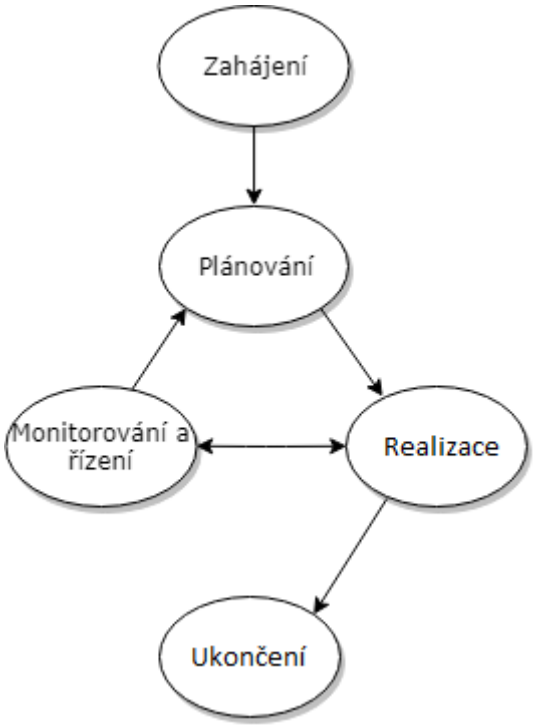
Opakem tradičního přístupu je agilní přístup. Ten je založen na průběžném upřesňování cíle, díky interakci se zákazníkem či s uživateli výsledků projektu. Tento proces se označuje jako získávání zpětné vazby. Reagováním na vstupy od zákazníka je projekt dynamicky upravován a proto je vhodný na takové projekty, kde dochází k vývoji produktu, tedy tehdy když nelze předem kvalitně popsat a naplánovat vše do podrobností a bez zpětné vazby. Agilní přístup se často používá ve vývoji software.

2.1 Fáze projektu

Projekt se může nacházet v různých fázích (procesních skupinách). Tyto fáze se mohou lišit použitou metodou na řízení projektu, ale víceméně je struktura a podoba fází velice podobná. Nejčastějšími fázemi jsou:

- Zahájení - Start projektu a vytyčení jeho mezí
- Plánování - Plánování částí projektu
- Realizace - Implementace rozplánované části
- Monitorování a řízení - Zpětná vazba od zákazníka, podpora
- Uzavření - Oficiální ukončení projektu

Projekt začíná zahájením a končí uzavřením. Fáze plánování, realizace a monitorování a řízení se mohou opakovat až do ukončení projektu. Diagram fází viz Obrázek 2.1. Vodopádový model má pak jednotlivé fáze za sebou tak, že přechod do další fáze je až tehdy, když je daná fáze kompletně ukončena.



Obrázek 2.1: Plánovací diagram

2.2 Softwarové chyby

Softwarové chyba (také nazývána jako „softwarový bug“) se dá definovat jako chyba, defekt, porucha či selhání v počítačovém programu nebo systému, která způsobuje vytváření nekorektních nebo neočekávaných výsledků a nebo nezamýšlené chování.

Softwarové chyby, jsou nedílnou součástí každého softwaru. Projekt větších rozměrů bez žádné softwarové chyby je velmi vzácný, tedy se dá předpokládat, že v nějaké fázi vývoje se objeví nějaký bug. Proces hledání a řešení bugů je nazýván jako „debugování“ a často používá techniky nebo nástroje pro určení bugu.

Bugy vznikají při realizaci daného projektu a jsou často nalezeny při testování daného řešení samotným vývojářem, nebo nástroji používanými pro překlad a vývoj. Další proces, při kterém je nalezení potenciálního problému nejvíce pravděpodobné, je takzvaný „code review“, tedy revizi kódu. Při tomto procesu se na cizí kód(vytvořený jiným vývojářem) podívá další vývojář a překontroluje kód, za účelem ověření funkcionality a nalezení potencionálních problémů, které původního autora kódu nemusely napadnout.

Dalším zdrojem odhalení bugů je testování kódu. Existuje spoustu druhů

technik testování. Jedním z hojně používaných druhů testů jsou takzvané jednotkové testy. Ty z pravidla píší samotní vývojáři a slouží k ověření funkcionality části kódu. Hlavním účelem těchto testů je odhalení funkční chyby v případě, že je nutné změnit část kódu. Tyto testy pak slouží pro zajištění stejné funkcionality i po změně. Dalšími testy jsou

V okamžiku kdy je aplikace dodána ve stavu, kdy je testovatelná (tedy je vyvinuta, nainstalovaná, spuštěná a přístupná) pak se obvykle jako první spouští Smoke testy. Ty mají za úkol ověřit, že aplikace je skutečně vhodná k testům. Obsah těchto testů se liší tester od testera, firma od firmy. Ale můžeme si zde uvést nejčastější vlastnosti těchto testů. Smoke testy se obvykle zaměřují pouze na hlavní funkce aplikace a to pouze v jejich pozitivním průběhu (více zde). Netestují se validace vstupů ani formáty výstupů. Smoke testy mohou být automatizované a to i v případě, kdy je zbytek testů prováděn manuálně. To je dáno právě tím, že rozsah těchto testů je podstatně menší a navíc také tím, že se zaměřují na funkčnosti, u nichž se nepředpokládají velké změny a proto není potřeba automatizovaný testovací skript často upravovat. Stručně řečeno - smoke testy jsou jedním z typů testů, kde se automatizace uplatňuje. Naproti tomu velice často jsou smoke testy prováděny také formou free testů.

Úspěšné provedení smoke testů obvykle slouží jako vstupní kritérium pro spuštění dalších fází testování. Před smoke testy mohou proběhnout ještě testy Instalační. Jejich úkolem je ověřit, že aplikace je (v dodané podobě) instalovatelná. Tedy že dodávka obsahuje použitelný instalační balík, který obsahuje všechny nezbytné součásti a že instalace z tohoto balíku proběhne bez chyb až do spustitelnosti aplikace. Instalační testy nejsou příliš často prováděny stejnými testery jako zbytek testů. Obvyklejší je, že je provádí osoba, která je zodpovědná za správu testovacího prostředí a tedy i instalaci aplikace. Záleží samozřejmě na rozsahu projektu.

Jakmile jsou smoke testy a případné instalační testy úspěšně provedeny, začíná hlavní fáze testování. Její průběh je závislý na struktuře testované aplikace. Tím chci říct, že čím je testovaná aplikace jednodušší, tím méně fází má její testování. Já tu ale uvedu obvyklý postup testů u aplikací složitých, kterých je dnes naprostá většina.

Většina dnešních aplikací je složena z menších částí, kterým můžeme říkat moduly nebo například komponenty. Při testování takové aplikace se postupuje od menších částí k velkým celkům. Jako první se tedy provádí testování komponent. Jak je zjevné, toto testování se zaměřuje na nejmenší funkční celky aplikace, které je možné testovat samostatně (neplést s Unit testy, které se mohou zaměřovat i na menší části samostatně netestovatelné a které jsou prováděné vývojáři). Myšlenka těchto testů je snad celkem jasná

- chyby komponent se nejsnáze nacházejí právě na úrovni komponent. Pokud se chyba komponenty projeví při jejím integraci do systému nebo dokonce až při integraci systému s dalším systémem, může být její odhalení podstatně složitější.

Přestože tu mluvíme o nejmenší samostatně testovatelné části aplikace, stejně se můžeme dostat do situace, kdy její testování není v této "solitérní"podobě není úplně snadné. Jde o to, že jednotlivé komponenty spolu samozřejmě v rámci aplikace komunikují a tedy mají vstupy z jiných komponent a i výstupy mohou být určeny pro další zpracování dalšími komponentami. Existují různé způsoby, jak toto obejít, jako jsou různé simulátory a nástroje. Vstup může být samozřejmě také simulován tak, že ho komponentě prostě "podstrčíte"a podobně.

Z vlastní zkušenosti musím říct, že fáze testování komponent se často spojuje s dalšími fázemi testování, především s fází integračního testování. Já sám tuto fázi označuji za testování vnitřní integrace. Jde totiž o testování správné komunikace jednotlivých komponent uvnitř aplikace. Bez ohledu na to, jakým konkrétním technickým řešením je tato integrace realizována, obsahem těchto testů je ověření, že jednotlivé komponenty si ve správný okamžik předávají správným způsobem zprávy, které mají správný obsah i formát.

Po otestování integrace přichází čas systémových testů. Ty se zaměřují na aplikaci jako celek v podobě, v jaké by ji měl používat zákazník. Zde se již ověřuje soulad reálného chování s chováním očekávaným, provádí se testování možných negativních průběhů, validují se výstupy a podobně. Pokud už žádná jiná fáze testování není použita, systémové testy jsou prováděny vždy (tedy za předpokladu, že se aplikace testuje).

Následovat mohou testy integrace aplikace s vnějšími systémy. Tady obvykle mluvím o integraci vnější. Tato fáze má samozřejmě smysl pouze tehdy, pokud k takové integraci dochází. Pokud ale ano, pak jde o fázi důležitou, která bývá někdy podceňovaná. Aplikace, která sama osobě pracuje dobře, ale nedokáže se domluvit se systémy, které pro svoje použití potřebuje, je pochopitelně těžko použitelná. Integrace přitom musí být správně navržena už při vytváření analýzy, podle které je aplikace vyvíjena. Už v této fázi vývoje softwaru by mělo dojít k testování a to testování dokumentace, aby se předešlo právě takovým chybám chybám. Ale o tom zase někdy jindy v rámci povídání o metodikách.

Integrační testování u této vnější integrace je poměrně náročné na přípravu. Především proto, že na těchto testech obvykle spolupracují všechny dotčené subjekty, tedy dodavatelé systémů, se kterými je testovaná aplikace integrována. Testování tak může mít podobu "štafetového závodu", kdy je-

den tester ve svém systému provede nějakou operaci a druhý tester v tom svém sleduje výsledek. Samozřejmě opět záleží na konkrétním technickém řešení integrace. Pro testování mohou být využívány simulátory, testování může být prováděno i s pomocí tzv. "fake"vstupů, tedy vstupů které se tváří jako reálné reakce integrovaného systému, ale jejich obsah je upraven pro potřeby testování. První fáze těchto integračních testů se navíc obvykle soustředí jen na ověření správné funkce rozhraní, které je k integraci určené. Tedy na formát a obsah zpráv, které aplikace na toto rozhraní posílá.

Po úspěšném průchodu všemi fázemi systémových a integračních testů je aplikace vhodná pro testy akceptační. Tedy za předpokladu, že výsledky těchto testů splňují kritéria dohodnutá pro přechod do další fáze. O těchto testech zase někdy příště. A koukám, že zase až tak stručný jsem nebyl.

TODO Pri jakzch fazich vznikaji bugy link2

TODO pridat vyznamnost bugu link3

TODO Moznosti zapisovani bugu - github, jira, bugzilla, redmine link4

TODO pridani vyhody trackovani- snadna detekce duplikatu, klasifikace - prideleni odpovednym oddelenim, klasifikace - nejpravdepodobnejsi pricina, prioritizace link5

TODO zakladni klasifikace chyb

3 Klasifikační metody

Literatura

[1] Hoare, C. A. R. Algorithm 64: Quicksort. *Commun. ACM*. July 1961, 4, 7, s. 321. ISSN 0001-0782. doi: 10.1145/366622.366644. Dostupné z: `http://doi.acm.org/10.1145/366622.366644`.

[2] *Class Graphics2D* [online]. Oracle, 2016. [cit. 2016/03/09]. Java SE Documentation. Dostupné z: `https://docs.oracle.com/javase/7/docs/api/java/awt/Graphics2D.html`.

[3] Knuth, D. E. *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. Addison-Wesley Longman Publishing Co., Inc., 1997. ISBN 0-201-89684-2.