Semestrální práce ke kurzu 4IT421 Zlepšování procesů budování IS	
Semestr	LS 2016/2017
Autoři - jméno, příjmení, xname	Martin Jonák, xjonm14 Daniel Šrám, srad01
Téma	Ways to Make Code Reviews More Effective
Datum odevzdání	14.5.2017

Abstrakt

Práce seznamuje čtenáře se způsoby, jak provádět revize kódu maximálně efektivně a tak, aby přinášelo největší užitek. Na úvod jsou vysvětleny základní principy a pojmy z oblasti revize kódu. Následuje objasnění výhod a nevýhod procesu revize kódu společně s možnostmi jeho automatizace. Na závěr jsou popsány nástroje, které lze využít pro podporu revizí kódu.

Klíčová slova

revize kódu, code review, automatické nástroje, best practices

Obsah

Úvod

V současné době vznikají aplikace jako výsledek práce mnoha lidí. Nejedná se jen o spolupracovníky pracující v jednom týmu, ale i o spolupráci více týmů. Každý programátor má svůj styl práce a své názory na řešení určitých problémů. Jak v takovém prostředí zajistit dostatečnou kvalitu výsledného produktu? Jak dohlédnout na dodržování standardů? Nestačí jen, že kód nějak pracuje, ale je nutné aby byly dodrženy i dané zásady a postupy. Například zda byli dodrženy určené návrhové vzory. Zda jsou ošetřeny potencionální slabá místa systému, které by mohli vést k zneužití jinak správně fungující aplikace. I aplikace která pracuje na první pohled správně, může obsahovat zásadní chyby, které výrazně ovlivňují její výkon. Taková aplikace také může obsahovat nedostatky, které se projeví teprve v budoucnu. Například při rozšiřování aplikace.

Dalším z problémů se kterým se při vývoji aplikací lze setkat je čitelnost kódu. Čitelnost kódu se stává velmi důležitou při spolupráci více programátorů. Díky ní se mohou noví členové vývojového týmu rychleji zapojit do práce. Pokud je kód dostatečně srozumitelný, není problém rychle nahradit jednoho člena týmu druhým. Také lze snadněji provádět úpravy kódu v budoucnu.

Další problémy mohou nastat, pokud se programátoři zaměří čistě na svojí část kódu a zcela ignorují ostatní části. V důsledku toho může docházet k nekompatibilitě jednotlivých modulů aplikace, velkým rozdílům v kvalitě kódu a celkové roztříštěnosti výsledné aplikace.

Možným řešením výše popsaných problémů může být revize kódu (v angličině code review). Základní myšlenku tohoto procesu lze popsat českým příslovím: Více očí více vidí. V té nejjednodušší formě lze revizi kódu provést tak, že požádáme kolegu o prohlédnutí si naší práce a vyjádření svého názoru. Ovšem takový jednoduchý postup může fungovat pouze u velmi malých týmů. U větších týmu je nutné zavést propracovanější postupy a procesy, které zajistí, že revize kódu nebude pouze ztrátou času a zdrojem problémů, ale přinese očekávané výsledky v podobě vyšší kvality kódu a zlepšení spolupráce mezi programátory a to vše při co nejmenších nákladech. Existuje více možných přístupů, jak provádět revize kódu a více různých nástrojů k podpoře a automatizaci tohoto procesu.

Cílem práce je seznámit s technikou code review a popsat různé formy, včetně těch automatizovaných. Navíc bude čtenář seznámen se způsoby, jak provádět code review maximálně efektivně a tak, aby přinášelo největší užitek. K dosažení tohoto cíle je v práci uvedeno dostatečné množství informací na to, aby čtenář porozuměl procesu code review a dokázal jej efektivně využít.

Co je to revize kódu

Posouzení kódu je v informatice proces systematického zkoumání zdrojového kódu. Jeho účelem je najít programátorské chyby, které autor kódu přehlédl v prvotní vývojové fázi, čímž se zvýší kvalita software. Posouzení mohou být dělána různými způsoby, například párovým programováním, neformální kontrolou nebo formální softwarovou inspekcí (i pomocí automatizovaných nástrojů).

Existuje více možností, jak provádět revize kódu. Od jednoduchých a neformálních po formální. Nejjednodušší a nejméně formální možností revize kódu je diskuze nad kódem s ostatními kolegy. Pokud ovšem mají revize kódu přinášet opravdové výsledky, je nutné zavést více propracované postupy. Další možností používanou v distribuovaných systémech správy verzí je “pull request”, jenž funguje tak, že než jsou nové změny zaneseny do hlavní větve v repositáři, musí je schválit určené osoby. Přitom mají možnost prohlédnout kód a v případě nedostatků ho vrátit k přepracování.

Peer review je méně formální způsob revize kódu, při kterém jeden programátor prochází kód druhého. Obvykle bez formalizace, bez velkých příprav či agendy. Slouží pro vzájemné pochopení problému či řešení, brainstorming atd.

Částečně formalizovaná revize (Walk Through) je způsob revize kódu, při kterém autor provádí posluchače testovaným dokumentem či kódem. Posluchači označují chyby a defekty v dokumentu či kódu a jejich řešení může být na místě diskutováno. Tato revize může mít i pouze jednoho posluchače. Málo formální revize jsou vhodnější pro projety vedené agilně. (Grossl, 2017)

Formální inspekce (Inspection) je revize plánovaná předem a má moderátora. Vše je vedeno písemně, existuje k ní i agenda a účastníci musí být připraveni. Výsledkem je formální výstup. Formální inspekce kódu byla popsána Michaelem Faganem již v roce 1976. Má sedm fází, jednotliví účastníci mají přidělené role a skládá se z několika schůzí. (Cohen, 2007)

Aby měla jakákoliv revize kódu smysl, je nutné zajistit, aby nalezené nedostatky nebyly ignorovány, ale náležitě opraveny. Revize kódu se může zabývat mnoha oblastmi. Od stylu psaní a způsobu formátování po bezpečnost, výkon a celkový návrh. Samostatnou kapitolou jsou automatické revize kódu. Lidská práce je nákladnější než práce počítače. Proto existují nástroje, které dohlíží na kvalitu kódu a upozorňují programátora na nedostatky již během jeho práce. Možnosti těchto nástrojů jsou ale omezené a nelze jimi zcela nahradit lidskou mysl.

Výhody revize kódu

Revize kódu umožňuje udržovat kód čitelný a srozumitelný. Už pouze vědomí že někdo bude po Vás kontrolovat Vaši práci, vede ke snaze psát kód svědomitě. Tedy samotné zavedení revizí kódu má preventivní účinek. Díky čitelnému a srozumitelnému kódu je snazší spolupráce a noví členové týmu snadněji a rychleji proniknou do vývoje aplikace. Zásadní výhodou je sdílení vědomostí mezi jednotlivými pracovníky. Díky revizím musí jednotliví členové týmu porozumět práci ostatních. To vede k možnosti jednotlivé členy zaměnit během vývoje. Také to vede k lepším odhadům, protože ti kteří je provádějí, mají celkově lepší přehled o složitosti a problémech všech částí aplikace.

Zavedení procesu code review má spoustu pozitivních dopadů. Prvním a jasnou výhodou je zkvalitnění zdrojového kódu, jenž vede k menšímu počtu chyb a snazší údržbě aplikace. Mimo to nutnost číst cizí kód vede k rovnoměrnému rozložení znalosti aplikace mezi programátory a tím pádem ke snadnějším úpravám v budoucnu, kdy například autor kódu ji v týmu nebude a někdo jiný bude muset aplikaci upravovat. Další výhodou je, že jak autor kódu, tak člověk provádějící revizi mají možnost učit se novým postupům a pohledům na problém. Kvalifikace pracovníků tak díky code review stoupá.

Velkým přínosem je možnost kontrolovat kód z pohledu, jenž stroj nedokáže posoudit. Mezi prvky, které dokáže odhalit pouze lidská revize kódu patří problémy v celkovém návrhu aplikace a jejím designu. Lze dohlédnout i na to, zda byla při vývoji nového kódu dodržena architektura aplikace, běžně využívané principy a popřípadě i jiné praktiky, které jsou v rámci podniku využívány. Lze posoudit, zda nejsou zbytečně řešeny problémy, které by se dali vyřešit použitím vhodných návrhových vzorů. A pokud jsou použity návrhové vzory, pak jestli jsou použity odpovídajícím způsobem.

Dále lze kontrolovat rozložení kódu do logických celků. Tedy aby kód byl rozdělen do komponent a modulů a nedocházelo k promíchání částí odpovědných za zcela odlišnou funkcionalitu. Díky tomu je v možné v budoucnu aplikaci snadněji spravovat a jednodušeji upravovat. Také to přispívá k celkové čitelnosti kódu. Člověk také může posoudit, zda implementovaná funkcionalita nebyla již zpracována někde jinde. Například zda není možné po úpravách použít kód ze staršího projektu a ušetřit tak čas při vývoji, nebo zda-li není napsaný kód zbytečně komplikovaný. Také možné odhalit nepotřebné funkce či jinak zbytečný kód, který je pouze zdrojem komplikací.

Člověk, na rozdíl od stroje, dokáže posoudit srozumitelnost a čitelnost kódu. Osoba kontrolující kód dokáže posoudit, zda názvy polí, proměnných, parametrů, metod, funkcí, tříd, rozhraní a dalších objektů dávají smysl a odpovídají opravdu tomu, co pojmenované objekty představují. Dokáži porozumět kódu, aniž bych potřeboval pomoc autora? Jsou vložené komentáře opravdu přínosné, nebo je autor napsal jen aby splnil povinnost? Jsou okomentované všechny části kódu? Jsou texty vyjímek a chybových hlášení srozumitelné? Také je možné kontrolovat kvalitu testů. Například zda testy mají nějaký význam a jaké pokrývají oblasti kódu. Jsou všechny testy správně popsané a odpovídá jejich popis skutečnosti? Jsou testovány všechny požadavky?

Revize kódu nemusí být prováděna pouze ve smyslu kontroly. Ale může být použita i zaučení nových členů týmu. Například méně zkušený pracovník může získat mnoho znalostí, když prochází kód seniorního programátora. Zároveň může tento nový pracovník přinést nové nápady a návrhy, jak kód vylepšit. To vede nejen k zapracování nového člena, ale i k přínosu v podobě nového pohledu na staré problémy.

Nelze sepsat dokonalý seznam všeho, co lze během vývoje kontrolovat, ale v rámci projektu je dobré si připravit checklist s tím, na co je dobré se zaměřit.

Nevýhody revize kódu

I přes všechny výhody revize kódu zmíněné v předešlé kapitole, i tato technika skýtá několik rizikových faktorů, jenž lze považovat za nevýhody. Nicméně tyto faktory představují skutečnou hrozbu pouze v případě, kdy je revize kódu prováděna špatně. V tomto kontextu lze za největší nevýhody považovat časovou náročnost, perzistenci skrytých chyb a lidské faktory spjaté s přijímáním a poskytováním zpětné vazby.

Vycházíme-li z faktu, že nejrozšířenější metodou code review je prezentace přírůstku kódu jeho autorem, jenž může být uskutečněna buď "přes rameno" mezi dvěma programátory, nebo poněkud formálněji pomocí schůze, objevíme hned několik nevýhod, které mají dopad především na efektivitu celého procesu. Hned první a znatelnou nevýhodou je časová náročnost, a to hlavně v porovnání s přidanou hodnotou, jenž si účastníci takovéto schůzky skutečně odnesou. Metoda code review, při níž má hlavní slovo autor samotného kódu, který posluchačům prezentuje svoji práci, totiž v praxi často připomíná spíše jakýsi schvalovací proces a vzdaluje se tak od cíle revize kódu, nehledě na to, že se takové schůze účastní hned několik programátorů, jejichž pracovní kapacita je v danou chvíli nevyužitá. Nevýhoda metody revize "přes rameno" je pak zřetelná, jelikož získané poznatky mají dopad zpravidla jen na jednoho člověka. Revize kódu pomocí prezentace má také dopad na hloubku, do které se revizor nad kódem zamýšlí, proto se při použití této metody často neodhalí na první pohled méně viditelné problémy. Navíc naplánovat schůzku všech vývojářů může být samo o sobě problémové, poněvadž jsou často umístění v rozdílných zemích a časových pásmech. *(Tom Greaves, 2009)*

Nicméně druhou a závažnější nevýhodou revize kódu, která na každém projektu představuje znatelné riziko, souvisí se schopností spolupracovníků přijímat a poskytovat zpětnou vazbu. Při poskytování zpětné vazby jsou totiž klíčovou dovedností komunikační schopnosti, které občas vývojářům scházejí a může tak vznikat spoustu nedorozumění a napjatá nálada v týmu. Někteří lidé mohou dokonce revizi kódu zneužívat k lynčování jiných pracovníků. (Carlton Jenke, 2008)

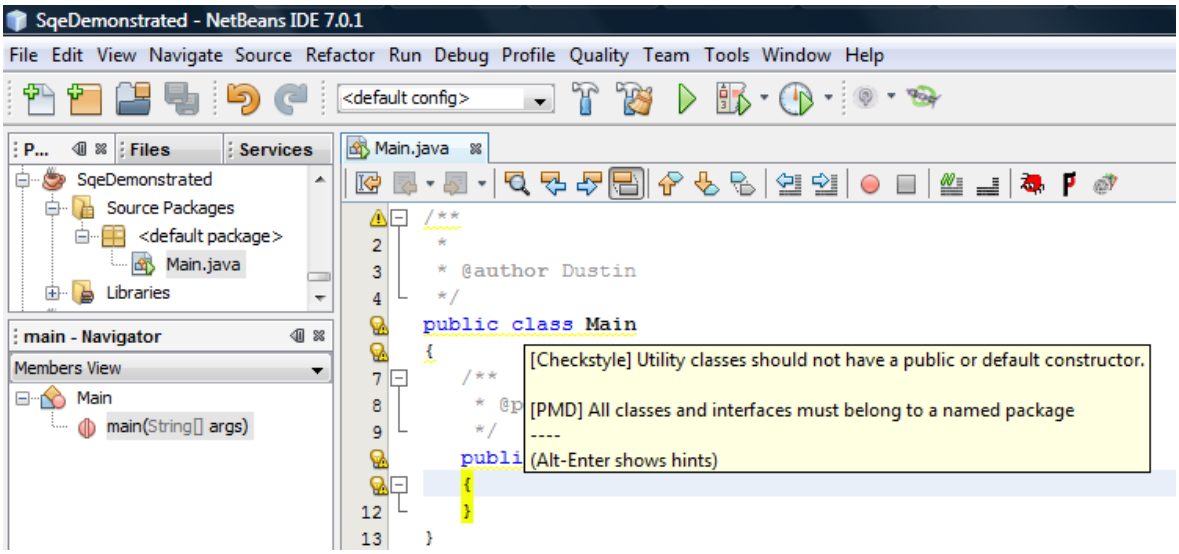
V této kapitole bylo uvedeno několik nevýhod revize kódu a čtenář by se mohl sám sebe ptát, zda-li je tedy vůbec vhodné code review provádět. Nicméně asi každý vývojář potvrdí, že revize kódu jsou vskutku velice přínosné, a to jak ze strany člověka, jenž provádí samotnou revizi, ze strany autora, tak se strany nezávislého pozorovatele. Některé nevýhody (jako například složité plánování společných schůzek) lze navíc eliminovat zavedením některých z automatizovaných nástrojů, jenž jsou popsány v následujících kapitolách. I přes všechny technické pomůcky zajišťující efektivitu a správnou formu revize je ovšem kriticky důležité, aby revizor ovládal techniku poskytování zpětné vazby a uměl správně komunikovat s ostatními lidmi.

Automatická revize kódu

Samostatnou kapitolou jsou automatické revize kódu. Lidská práce je nákladnější než práce počítače. Proto existují nástroje, které dohlíží na kvalitu kódu a upozorňují programátora na nedostatky již během jeho práce. Možnosti těchto nástrojů jsou ale omezené a nemohou zcela nahradit člověka. Co ale stroj dokáže, je kontrola formátování kódu a automaticky kód formátovat dle zadaných pravidel. Příkladem může být standart pro psaní kódu v jazyce PHP, PSR-2 (http://www.php-fig.org/psr/psr-2/) a nastavení do vývojového prostředí NetBeans, které automaticky upravuje kód dle daných pravidel (https://maniaplanet.github.io/netbeans-psr/).

Dalším možným nástrojem k revizi kódu je například PMD, který slouží k analýze kódu a upozorňuje na možné nedostatky. Ty hledá na základě připravené sady pravidel. Tento nástroj lze použít v jazycích Java, JavaScript, Salesforce.com Apex, PLSQL, Apache Velocity, XML či například XSL. Je možné jej integrovat do vývojových prostředí jako je Eclipse, NetBeans, JBuilder, JDeveloper a IntelliJ IDEA. Ukázku z prostředí NetBeans lze vidět na obrázku 1.

Zmíněné nástroje dokáží velmi rychle odhalit velké množství nedostatků. Na rozdíl od programátora nehrozí, že by něco přehlédli nebo vynechali. Díky neustále doplňovaným sadám pravidel v sobě udržují velké množství znalostí a pro určitou oblast revize kódu jsou velkým přínosem. Ale mají své limity. Automatický nástroj například dokáže rozpoznat, zda u metody je zapsán komentář, ale nedokáže již vyhodnotit jeho správnost a čitelnost. Dokáže říct, že kód aplikace je zapsán správně, ale nedokáže posoudit, zda aplikace odpovídá daným požadavkům. Automatizace se hodí tam, kde lze nedostatky popsat pravidly. Zatímco lidské zdroje je nutné nasadit u problémů, které vyžadují pochopení a porozumění kódu.

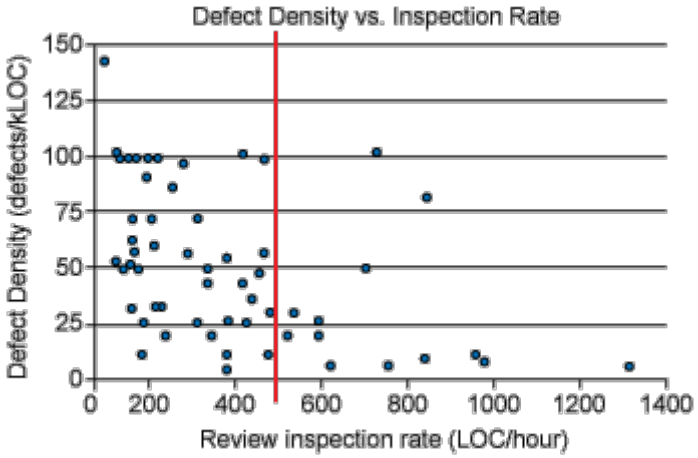


Obrázek 1: Nástroj PMD v prostředí NetBeans.

Best practices pro revize kódu

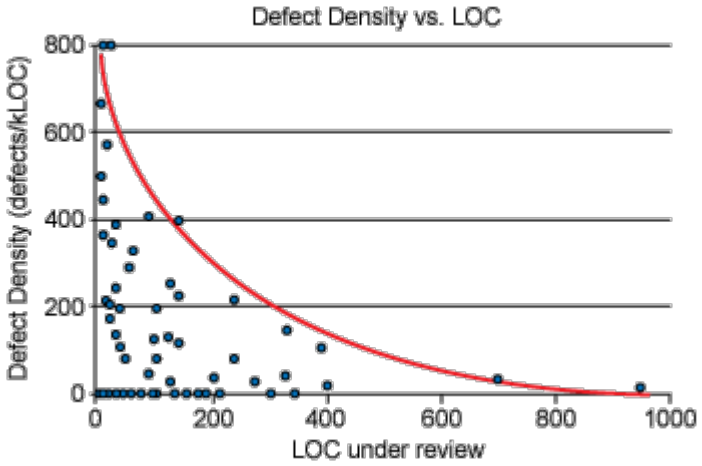
Úspěšná revize kódu vyžaduje nalezení rovnováhy mezi striktně zdkokumentovanými procesy a přátelskou týmovou náladou. Jak již bylo v této semestrální práci zmíněno, code review může mít i neblahé dopady, pokud není prováděna dobře. Proto by proces kontrolování kódu neměl být zatížen přílišnou disciplínou a manažeři a týmoví vůdci by se měli snažit nalézt zmíněnou rovnováhu a docílit tak časově efektivní revize kódu, jenž vzdělává zúčastněné. Následující tipy by měli zajistit efektivní code review.

Studie provedené společností Cisco Systems odhalila, že efektivní code review by neměla zahrnovat více než čtyři sta řádků kódu (anglicky *lines of code* - LOC). V některých případech by dokonce neměla přesahovat ani 200 LOC. Studie toto tvrzení argumentuje tím, že lidský mozek není schopen větší objem informací pojmout a úspěšně analyzovat. Při překročení 400 LOC se již snižuje pravděpodobnost odhalení defektů aplikace. Jak lze vidět na následujícím grafu, praxe odhaluje, že dvě stě až čtyři sta řádků kódu by mělo odkrýt sedmdesát až devadesát procent skrytých chyb. Pokud tedy revizor kontroluje kód, ve kterém se nachází deset defektů, správně provedená revize by měla odhalit sedm až devět chyb. (Smartbear.com, 2017)



Obrázek 2: Graf počtu chyb a rychlosti revize

Následující graf znázorňuje závislost počtu nalezených chyb v závislosti na počtu řádků zdrojového kódu v revizi. Na tomto grafu lze názorně vidět efektivní hranici objemu kontrolovaného kódu, jenž se pohybuje do 400 LOC.



Obrázek 3: Graf počtu chyb a objemu kódu

Další obecně platné doporučení se týká rychlosti revize. Abychom docílili výše uvedené míry odhalení chyb, neměli bychom za jednu hodinu kontrolovat více než pět set řádků kódu a zároveň by taková revize (ať už individuální revize nebo schůzka) svou délku neměla pokořit právě jednu hodinu. Ačkoli se může zdát velmi lákavé proskočit revizí doufajíc, že chyby eventuálně odhalí někdo jiný, nicméně praxe ukazuje, že při překročení pěti set LOC za hodinu znatelně klesá míra odhalení chyb. Stejně jako by revizor neměl kontrolovat kód příliš rychle, neměl by ani kontrolovat příliš dlouho. Pokud se člověk dlouho zamýšlí nad nějakým problémem a nedopřává si u toho pravidelné pauzy, jeho efektivita práce po jedné hodině klesá. Revize kódu s přiměřenou dávkou kódu a v přiměřeném, limitovaném čase přináší nejlepší výsledky.

Neméně důležitým doporučením pro kvalitní revizi kódu je nastavení cílů kontroly a společných metrik. Samotné implementaci procesu zajišťujícího kvalitu kódu by mělo předcházet vybrání reálných cílů a způsobů, jak bude jejich dosažení měřeno. Ke stanovení zmíněného cíle by měla ideálně být použita nějaká metoda na správné stanovení cílů, jako je například metoda SMART (Specific - konkrétní, Measurable - měřitelný, Achievable - dosažitelný, Realistic - realistický, Time-bound - ohraničený v čase). Mimo cílu stanoveného touto metodou, jenž může být například “Snížení počtu chyb z vývojové fáze na polovinu”, je dobré sledovat i interní metriky spojené s kontrolou kódu. Mezi tyto míry patří rychlost revize kódu, počet odhalených chyb za jednu hodinu revize a míra nalezených chyb na jeden řádek kódu. Je ovšem nutné zmínit, že uvedené metriky lze reálně sledovat pouze v případě automatizovaných nebo striktně popsaných code review procesů. Na druhou stranu code review řízené metriky trochu eliminuje lidský faktor a může předcházet interpersonálním problémům souvisejících s poskytováním a přijímáním zpětné vazby.

Následující tip míří převážně na autory kódu. Ti by totiž měli svůj kód komentovat. Okomentovaný kód provází revizora změnami a nabízí hlubší proniknutí do celého kontextu problému. Komentáře by měli cílit na člověka, který kód revizuje. Za přidanou hodnotu komentování kódu lze považovat fakt, že autor často nalezne mnoho nedostatků své práce již při komentování. Tím pádem se do revize dostane méně chyb a sníží se i míra chybovosti kódu.

Jak již bylo zmíněno, některým lidem může být nepříjemné, když jejich práci někdo kontroluje. A pokud dokonce dojde i k situaci, že revizující osoba nalezne nějaké nedostatky, autor onoho nedostatku může popsanou situaci nést špatně. Tomuto fenoménu lze předcházet vytvořením pozitivní atmosféry v týmu. Tato atmosféra by měla mířit hlavně na vzdělávací efekt revize kódu a spíše odstínit fakt, že management posuzuje kvalitu jednotlivých pracovníků i na základě nastavených metrik, jejichž dodržování si kontrolují pracovníci navzájem. Důležité také je porozumění týmu, že ne každá nalezená chyba je čistě negativní věc, ale lze ji využít na zlepšení kvality kódu celého týmu jako celku.

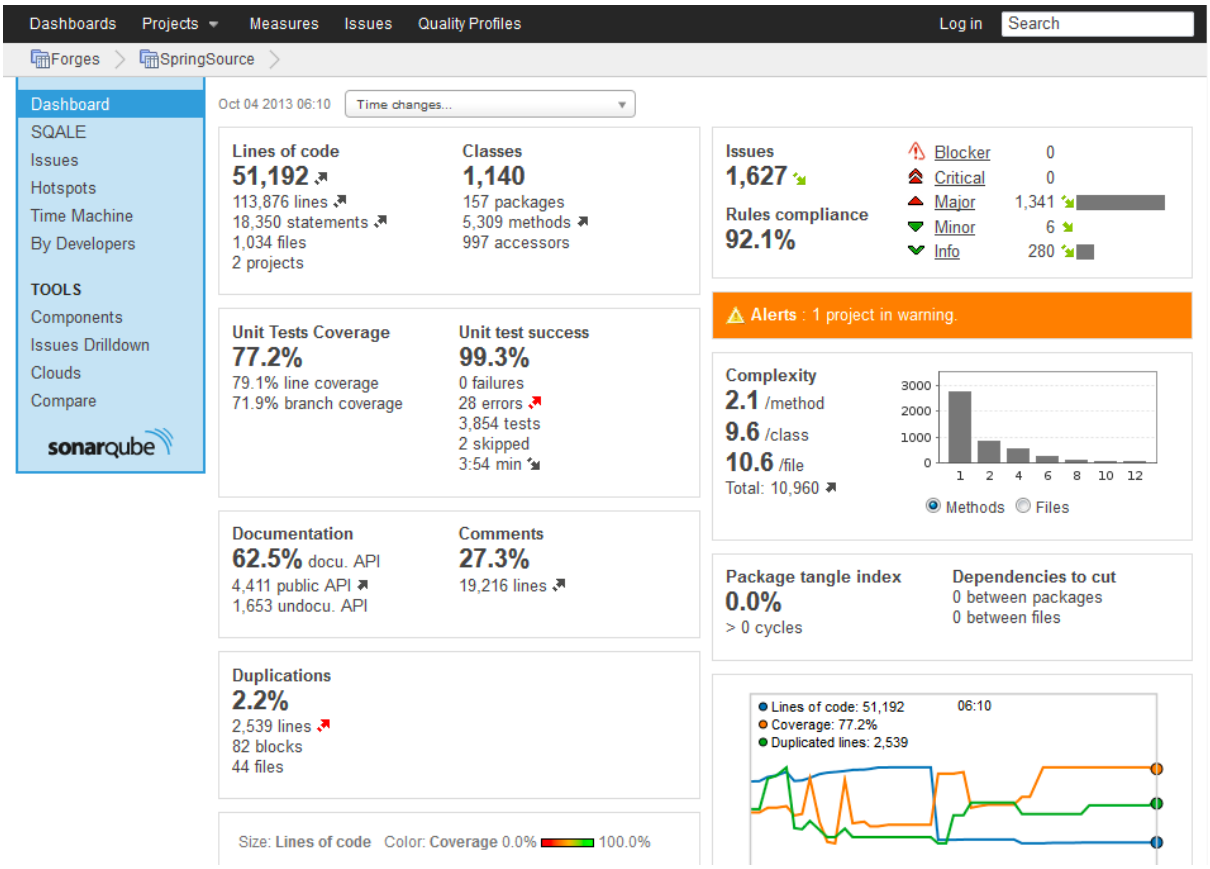
I přes optimalizaci procesu code review z časového pohledu, z pohledu objemu kontrolovaného kódu a pomocí vytvoření společných metrik, stále zůstává otázka - jakým způsobem budou nalezené chyby opravovány? Podle průzkumů společnosti Cisco Systems tuto otázku mnoho pracovních týmů stále nemá zodpovězenou. Nejlepším způsobem opravy nalezených chyb je použitím některého nástroje pro podporu revize kódu. Přehled v současnosti nejpopulárnějších a nejpoužívanějších nástrojů lze nalézt v následující kapitole.

Nástroje pro podporu revize kódu

V dnešní době existuje již mnoho nástrojů na podporu revize kódu. Tyto nástroje lze rozdělit na programy zajišťující statickou kontrolu kódu již při jeho psaní a překladu (tzv. *pre-commit* nástroje - čili nástroje používané před zveřejněním kódu) a na nástroje ulehčující samotnou revizi či nástroje poskytující již zmíněné statistiky a metriky (tzv. *post-commit* nástroje). Nástroje pro podporu samotné revize jsou často přizpůsobené konkrétnímu systému používánému pro sdílení a ukládání zdrojového kódu. Zabývat se budeme především nástroji podporující verzovací systém Git (v britském slangu výraz pro nepříjemného či protivného člověka), nicméně převážná většina z nich lze v nějaké omezené podobě využít pro dnes již překonaný systém subverzí (SVN, CVS).

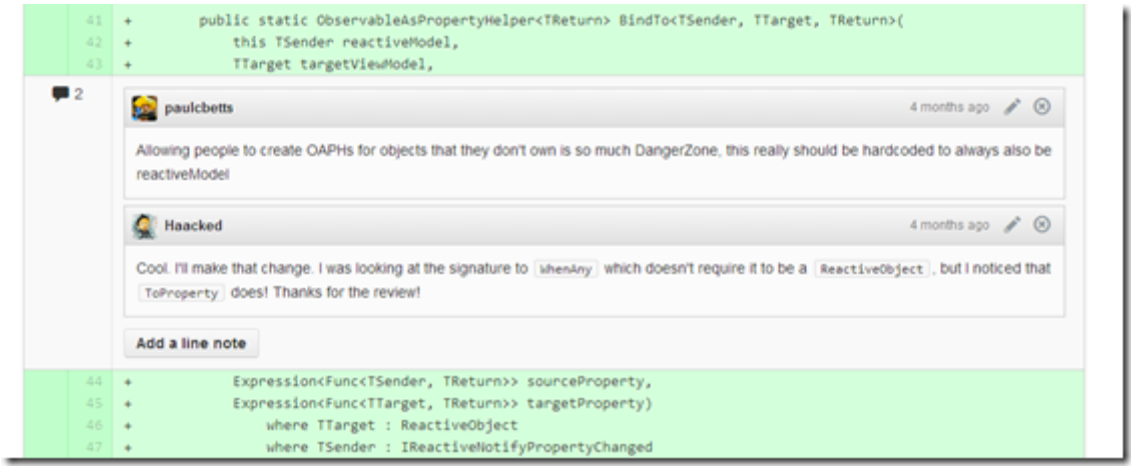
Uveďme nejprve základní přehled nástrojů pro statickou kontrolu kódu. Užití analytických metod k odhalení potencionálních chyb v kódu již při jeho tvorbě je jeden ze základních pilířů vývoje softwaru. Programátoři se dopouští drobných chyb velice často, ať už se jedná o chybějící středník, zbytečný parametr nebo duplicitní kód, nástroje zabudované přímo ve vývojovém prostředí to včas odhalí a autora upozorní. Popisované nástroje se nazývají lintery a jsou k mání pro leckterý jazyk (nejen programovací). Zajímavostí může být fakt, že první linter vznikl ve firmě Nokia jako derivát z kompilátoru pro jazyk C. Dnešní lintery již umí odhalovat i komplexní problémy od chybějící implementace rozhraní po cyklické závislosti. K tématu nástrojů pro statickou revizi kódu lze ještě uvést sdílené konfigurace formátování kódu, které jsou podporované napříč vývojovými prostředími a zajišťují jednotný formát zdrojového kódu. Statickou analýzu kódu používají i nástroje pro kontinuální integraci. Příkladem takového programu může být například nástroj Jenkins. (An overview of the Static Code Analysis approach in Software Development, 2009)

Co se týče *post-commit* nástrojů statické analýzy kódu, za zmínku stojí například nástroj SonarQube. Ten analyzuje kód uložený ve verzovacím systému a poskytuje zajímavé statistiky, jenž lze použít pro zhodnocení kvality kódu, viz. následující obrázek.



Obrázek 4: SonarQube (http://nemo.sonarqube.org)

Post-commit nástroje ale nabízejí mnohem více než statickou analýzu kódu. Mimo to totiž navíc podporují proces code review a mohou složit i jako nástroj pro komunikaci mezi programátory a v případě rozsáhlých open-source projektů je to často dokonce i jediný nástroj komunikace. Klasickým případem podpory procesu code review je například *pull request* (požadavek na revizi kódu před jeho začleněním do zbytku aplikace), jenž dnes nabízené nástroje graficky reprezentují. Revizor tak může komentovat cizí kód, psát své výtky či pochvaly k jeho jednotlivým částem, schvalovat začlenění kódu či zdrojový kód vrátit autorovi k opravě. Jedním z nejznámějších nástrojů je například GitHub, jenž nabízí jak desktopovou aplikaci, tak webové rozhraní. Ukázka grafické reprezentace usnadňující revizi kódu je k nahlédnutí na následujícím obrázku.



Obrázek 5: GitHub (zdroj vlastní)

Nástroje podobného charakteru jsou k dostání ať už v komerční nebo neplacené verzi od několika firem. Příkladem může být třeba Codacy, Crucible od společnosti Atlassian, jenž podporuje i staré verzovací systémy, nebo například GitLab.

Závěr

V dnešním světe, kde je téměř vše řízeno nějakým IT systémem, je dodržování určité štábní kultury zdrojového kódu rozhodně na místě. V této semestrální práci byla revize kódu podrobně rozebrána z několika úhlů pohledu. Čtenář měl možnost odhalit, co to revize kódu vlastně je, jaké nabízí výhody a jaká skrývá rizika. Práce dále odhaluje automatické nástroje v oblasti code review, jenž jsou nedílnou součástí kvalitního kódu, ale zároveň zmíněná kapitola uvádí i jejich nedostatky, kterých by si uživatelé měli být vědomi.

Aby byla práce vskutku kompletní a téma revizí kódu ucelené, byly v práci zmíněny i best practices pro revize kódu. V této byl čtenář obohacen o cenné tipy jak pro autora kódu, tak pro osobu provádějící revizi. V rámci toho bylo doporučeno i použití některého nástroje pro podporu procesu revize a tyto nástroje byly dále rozebrány v následující kapitole.

Pokud chceme provádět revize kódu smysluplně, měli bychom dodržovat určitá pravidla. Ta je nutné přizpůsobit velikosti týmu a zavedeným zvyklostem ve firmě. Při zavádění takových pravidel si ovšem musíme položit několik otázek. Například kdo bude kontrolovat čí kód? Kdo za co zodpovídá? Kdy bude kontrola prováděna? Jakou formou bude prováděna? Co je vhodné automatizovat a co naopak vyžaduje lidské oko? Na všechny uvedené otázky čtenář v semestrální práci našel odpověď a cíl práce tak byl splněn.

Zdroje

Tom Greaves (2009). Disadvantages of Code Review Walk-Through [online]. Dostupné z: http://ezinearticles.com/?Disadvantages-of-Code-Review-Walk-Through&id=2485233 [22. 3. 2017].

Carlton Jenke (2008). What are the pros and cons of design/code reviews? [online]. Dostupné z: http://stackoverflow.com/questions/182738/what-are-the-pros-and-cons-of-design-code-reviews [22. 3. 2017]

Smartbear.com. (2017). *Best Practices for Code Review* | *Learn Code Review*. [online] Dostupné z: https://smartbear.com/learn/code-review/best-practices-for-peer-code-review/ [8. březen 2017]

En.wikipedia.org. (2017). *Code review*. [online] Dostupné z: https://en.wikipedia.org/wiki/Code_review [8. březen 2017].

Cohen Jason (?). Four Ways to a Practical Code Review [online]. Dostupné z: http://www.methodsandtools.com/archive/archive.php?id=66 [28. 3. 2017]

Zdeněk Grossl (2017). Příprava návrhu testů,Statické a dynamické testy [online]. Materiály předmětu 4IT474 Analýza a návrh testů softwaru [28. 3. 2017]

Gomes, Ivo; Morgado, Pedro; Gomes, Tiago; Moreira, Rodrigo (2009). An overview of the Static Code Analysis approach in Software Development (PDF). Universidade do Porto.[8. březen 2017].

Gee, T. (2016). Ways to Make Code Reviews More Effective. [online] InfoQ. Dostupné z: https://www.infoq.com/articles/effective-code-reviews?utm_source=infoqWeeklyNewsletter&utm_medium=WeeklyNL_EditorialContent_culture-methods&utm_campaign=10042016news [8. březen 2017].