



KOMPUTEROWE PRZETWARZANIE WIEDZY

**Kolekcja prac 2012/2013
pod redakcją Tomasza Kubika**



SPIS TREŚCI

1	Audyt w systemach informatycznych	3
1.1.	Dostępne systemy pomiaru pracy	3
1.1.1.	Systemy kontroli wersji	3
1.1.2.	Systemy zarządzania projektem	4
1.1.3.	Systemy do statycznej analizy kodu	5
1.2.	Sposoby pomiaru jakości pracy programisty	6
1.2.1.	Ocena jakości wygenerowanego kodu	6
1.2.2.	Ocena efektywności pracy	8
1.3.	Proponowane rozwiązanie	9
1.3.1.	Eclipse IDE	9
1.3.2.	Eclipse PDE	9
1.3.3.	Zaimplementowane kryteria oceny kodu	9
1.3.4.	Utworzona wtyczka	10
	Bibliografia	12

AUDYT W SYSTEMACH INFORMATYCZNYCH

M. Nowak, G. Maj

Jednym z podstawowych problemów profesjonalnego zarządzania firmą w branży informatycznej jest zarządzanie jakością. Jakość można definiować i mierzyć na wielu poziomach. Bardzo dużo uwagi poświęcane jest jakości produktów, choć równie ważnym elementem jest ocena jakości pracy pracowników oraz ich efektywności. Audyt w systemach informatycznych powinien zawierać także elementy oceny jakości pracy i służyć usprawnieniu działania organizacji.

W rozważaniach na temat oceny jakości pracy i wydajności pracowników skupiono się na firmach informatycznych, których przedmiotem działalności jest wytwarzanie oprogramowania. Kluczowym pracownikiem biorącym udział w tworzeniu końcowego produktu, czyli programu jest programista. Ocena pracy programisty jest zadaniem złożonym i wymaga znajomości wielu aspektów procesu wytwarzania oprogramowania. W dzisiejszych czasach zarządzanie jakością oprogramowania jest prawdziwym wyzwaniem dla dużych firm o zasięgu globalnym zatrudniających tysiące programistów z całego świata.

Podstawowym pytaniem, które postawiono w tym rozdziale jest: jak efektywnie mierzyć jakość i efektywność pracy programisty? W niniejszym artykule autorzy postarają się przybliżyć problem, przedstawić dostępne na rynku rozwiązania oraz opisać prosty system kontroli efektywności pracy.

1.1. Dostępne systemy pomiaru pracy

Na rynku istnieją różne systemy służące do pomiaru i wspomagania pracy programisty. Skupiają się one głównie na łączeniu kodu i kontroli pracy wielu programistów, pracujących nad jednym projektem. Poniżej przedstawione zostały najpopularniejsze systemy, w tym rozbudowane i szeroko wykorzystywane systemy otwartoźródłowe.

1.1.1. Systemy kontroli wersji

Systemy kontroli wersji stworzone są do pomocy przy rozwoju oprogramowania, ich głównym zadaniem jest łączenie pracy wielu pracowników jednego pro-

1. Audyt w systemach informatycznych

jektu, jednak poprzez tworzenie historii zmian w projekcie daje możliwość kontroli i oceny pracy pracowników. Dają one możliwość śledzenia postępu prac poczynionych przez poszczególnych pracowników. Podstawowe informacje, które można uzyskać to:

- jaki użytkownik i kiedy wprowadzał zmiany,
- możliwość sprawdzenia jakie zmiany zostały wprowadzone w danym commitcie:
 - dokładne różnice między zmienionymi plikami,
 - skrócone statystyki (ilość zmian bez poszczególnych różnic),
 - sumy kontrolne poszczególnych zmian,
 - wyświetlanie tylko informacji przez nas określonych,
- możliwość dostosowania informacji przez nas śledzonej:
 - kontrola jednego pracownika,
 - kontrola danego okresu zmian,
 - kontrola historii tworzenia i łączenia gałęzi.

Przykładami systemów kontroli wersji są najbardziej popularne darmowe systemy takie jak: Git [1], Svn [2] czy Mercurial [3].

Git

Git jest rozproszonym systemem kontroli wersji. Jest to wolne oprogramowanie udostępnione na licencji GNU GPL2. Git powstał jako alternatywa dla zamkniętego systemu BitKeeper w kwietniu 2005 roku do rozwijania projektu Linux.

Svn

Subversion jest wolnym systemem kontroli wersji udostępnionym na licencji Apache. Został stworzony do zastąpienia systemu CVS.

Mercurial

Mercurial podobnie jak Git jest rozproszonym systemem kontroli wersji. Został napisany w tym samym czasie jak Git, również w celu pomocy do rozwijania projektu Linux. Jest wykorzystywany w wielu projektach otwartoźródłowych.

1.1.2. Systemy zarządzania projektem

Systemy zarządzania projektem są platformami wspomagającymi podział prac i weryfikację postępów prac. Z tego tytułu dają pewien obraz pracy włożonej przez każdego współpracownika. Poniżej zostały przedstawione najpopularniejsze systemy zarządzania:

- **Redmine**
Redmine [4] jest wolnym, otwartoźródłowym internetowym narzędziem do zarządzania projektem i szukania błędów. Udostępnia kalendarz i wykres Ganta

1.1. Dostępne systemy pomiaru pracy

wraz ze wsparciem wizualnym by łatwiej kontrolować kończące się etapy projektu. Wspiera różne systemy kontroli wersji i umożliwia kontrolę nad podziałem zadań w grupie. Narzędzie Redmine jest oparte o framework Ruby on Rails, co zapewnia mu przenośność.

- **Jira**

JIRA [5] jest zamkniętym oprogramowaniem stworzonym przez firmę Atlassian, które ma na celu wspomaganie w zarządzaniu projektem i śledzeniu błędów. Jest wykorzystywana w wielu projektach m.in. Fedora Commons, Skype czy JBoss. Jest narzędziem wielopatformowym dzięki oparciu go o język Java. Firma Atlassian udostępnia program JIRA nieodpłatnie do użytku non-profit.

- **GitHub**

GitHub [6] jest serwisem hostingowym dla projektów wykorzystujących system kontroli wersji Git. Udostępnia darmowe otwarte repozytoria a także płatne repozytoria zamknięte. Oprócz podstawowego zadania jakim jest przechowywanie plików projektu wspomaga kontrolę nad tworzeniem oprogramowania poprzez dodatkowe narzędzie nieobecne w systemie Git np. bugtracker, forki repozytoriów, graficzne statystyki czy wiki z dokumentacją. Umożliwia łączenie programistów w organizacje i przydzielanie możliwości rozwijania oprogramowania dla konkretnych użytkowników.

1.1.3. Systemy do statycznej analizy kodu

Platformami najbardziej zbliżonymi do zmian w samym kodzie projektu, a nie tylko dużymi zmianami w strukturze całego projektu, bądź gotowymi działającymi zmianami obserwowanymi poprzez zmianę plików w repozytorium, są systemy do statycznej analizy kodu. Ich cechą jest praca równolegle do tworzonego kodu, a nie obserwacja zmian etapami. Obecnie są powszechnie dostępne platformy obsługujące wszystkie popularne języki programowania. Przykładami takich systemów są:

- **Klocwork**

Klocwork [7] jest zamkniętym programem przeznaczonym do statycznej analizy kodu pozwalającym na detekcję błędów strukturalnych kodu i błędów przebiegu programu. Umożliwia pracę z najpopularniejszymi językami C, C++, Java czy C#.

- **Pylint**

Pylint [8] jest otwertoźródłowym oprogramowaniem do znajdowania błędów w kodzie i sprawdzania jego jakości. Pylint jest podobny do innego oprogramowania Pychecker ale jest bardziej rozbudowany. Wspiera między innymi sprawdzanie długości linii, sprawdzanie zgodności nazw zmiennych ze standardem kodowania czy tworzenie diagramów UML ze stworzonego kodu.

- **CodeSonar**

CodeSonar [9] jest zamkniętym oprogramowaniem analizującym kod pod względem bezpieczeństwa i poprawności. Wspiera oprogramowanie napisane w C, C++ i Java. Jest wykorzystywany w wielu gałęziach gospodarki

1. Audyt w systemach informatycznych

m.in. w NASA czy w firmie Toyota. Oprócz analizy kody umożliwia wizualizację architektury programu i zbiera dane do metryk oceny jakości kodu.

1.2. Sposoby pomiaru jakości pracy programisty

Na ocenę pracownika programisty można patrzeć z różnych perspektyw. Z jednej strony, efektem jego pracy jest kod źródłowy, więc podstawowym składnikiem oceny powinna być jego jakość. Z drugiej strony jednak, w każdym projekcie obecne są pewne ograniczenia czasowe, więc szybkość pracy pracownika jest również ważnym aspektem oceny. W ogólności można powiedzieć, że jakość pracy programisty można sprowadzić do dwóch składników:

- oceny jakości kodu,
- oceny organizacji pracy.

Normy dotyczące zarządzania jakością (np. ISO 9000:2008, ISO 15504-4:2005) definiują tzw. kluczowe wskaźniki efektywności (ang. KPI - key performance indicator). Ważną cechą tych wskaźników jest to, że opisują one mierzalne procesy w postaci liczbowej. Daje to menedżerowi obiektywny obraz efektywności danego procesu w przedsiębiorstwie i pozwala na cykliczną kontrolę jakości. Poniżej przedstawiono kilka przykładowych wskaźników, które spełniają warunek mierzalności, więc kwalifikują się do grupy KPI.

1.2.1. Ocena jakości wygenerowanego kodu

Jakość kodu jest bardzo szerokim zagadnieniem, które po wielu dekadach intensywnego rozwoju branży IT nie zostało jeszcze dostatecznie zbadane. Wynika to stąd, że na przestrzeni lat zmieniały się trendy w tworzeniu oprogramowania – programiści stosowali różne paradygmaty programowania (w kolejności chronologicznej: programowanie obiektowe, programowanie strukturalne, programowanie funkcjonalne). Dziś dominującym paradygmatem jest programowanie obiektowe, lecz wydaje się, że prawdziwa jest teza, że każda z wymienionych filozofii tworzenia oprogramowania jest użyteczna dla pewnych zastosowań.

Stąd obiektywny pomiar jakościowy kody jest trudny. Niektóre elementy języka mogą być uznawane za wartościowe w jednym podejściu, a drugim absolutnie niedopuszczalne. Istnieją jednak pewne fundamentalne cechy kodu, które są pożądane zawsze, np. zwięzłość kodu.

Wartości służące do pomiaru pewnej własności oprogramowania lub jego specyfikacji nazywane są metrykami oprogramowania. Aby metryka była użyteczna powinna być:

- prosta i możliwa do obliczenia przez komputer,
- przekonująca,
- konsekwentna i obiektywna,
- spójna pod względem użytych jednostek,
- niezależna od języka oprogramowania,
- dająca przydatne informacje [10].

1.2. Sposoby pomiaru jakości pracy programisty

Metryki można podzielić na statyczne i dynamiczne. Metryki statyczne łączą się ściśle z analizą statyczną kodu, dziedziną inżynierii oprogramowania zajmującą się badaniem struktury kodu źródłowego. Metryki te najbardziej przydatne są dla samych programistów i innych osób bezpośrednio zaangażowanych w proces powstawania oprogramowania. Pozwalają na bieżące śledzenie jakości kodu i zwrócenie uwagi na miejsca, które wymagają uproszczenia bądź szczególnie uważnego testowania. Niemniej jednak, programista, który dostarcza kod o dobrych statycznych metrykach może być uznawany za dobrego pracownika.

Metryki dynamiczne to wskaźniki abstrahujące od kodu źródłowego. Badają one zachowanie programu po jego uruchomieniu. Są one mocno związane z wymaganiami klienta, stąd dla samego przedsiębiorcy, są to liczby przydatne bardziej do analizy jakości produktu niż pracy jego pracowników.

Linie kodu (LOC)

Najprostszą metryką rozmiaru oprogramowania jest liczba linii kodu źródłowego. Mimo jej prostoty, wymaga ona pewnego rozróżnienia, bowiem programy pisane w językach wysokiego poziomu mają znacząco mniej linii kody niż programy pisane np. w assemblerze. Do tej metryki z reguły nie są wliczane również linie zawierające tylko komentarze.

Ważnym aspektem jest tutaj również styl pisania kodu. Poniższe dwa przykłady obrazują dwa różne style programowania w języku C++ obrazujące ten niuans:

```
// przykład 1
void bubblesort(std::vector<int>& A)
{
    int temp = 0;

    for (int i = 0; i < A.length() - 1; i++)
    {
        for (int j = 0; j < A.length() - 1 - i; j++)
        {
            if (A[j] > A[j+1])
            {
                temp = A[j+1];
                A[j+1] = A[j];
                A[j] = temp;
            }
        }
    }
}

// przykład 2
void bubblesort(std::vector<int>& A) {
    for (int i = 0; i < A.length() - 1; i++)
        for (int j = 0; j < A.length() - 1 - i; j++)
```

1. Audyt w systemach informatycznych

```
        if (A[j] > A[j+1])
            std::swap(A[j], A[j+1]);
    }
```

Oba przykłady są identyczną implementacją popularnego algorytmu sortowania bąbelkowego. W pierwszym przykładzie stosowane są przeniesienia do następnej linii przy otwieraniu nowych bloków kodu (klamr {}). Oprócz tego, każde wyrażenie warunkowe i pętle są akcentowane nowym otwarciem i zamknięciem klamry, choć składnia języka C++ tego nie wymaga. Dodatkowo, w drugim przykładzie, aby bardziej zwiększyć zwięzłość (i czytelność) kodu, do operacji zamiany wartości dwóch elementów wektora, zastosowano funkcję z biblioteki standardowej, która ma dokładnie takie samo działanie jak kod z przykładu pierwszego.

Linie kodu na plik źródłowy (LOCPF)

Kolejną metryką związaną z ilością linii kodu w plikach jest miara ilości linii kodu na plik. Metryka ta jest szczególnie ważna ze względu na to, że zapewnia ewaluację projektu programistycznego jako całości. Ważne jest, aby nie wliczać do tej metryki plików konfiguracyjnych ani plików wynikowych budowania.

Do wyliczenia LOCPF można wykorzystać metrykę LOC:

$$\frac{1}{N} \sum_{f=1}^N LOC(file[f]) \quad f: file[f] \in source_files$$

Brak spójności metod (LCOM)

Metryka ta bazuje na tezie, że wszystkie metody powinny być spójne. Stąd nazwa LCOM (*Lack of Cohesion of Methods*). Spójność metody oznacza że ma ona jedną kompetencję i nie wchodzi w kompetencje innych metod. Tak organizowany kod jest dobrze zrefaktoryzowany, a przez to łatwiej utrzymywalny, testowalny oraz czytelny.

Spójność metod (a także klas i innych obiektów języka) jest cechą, o której mówi jedna z zasad architektury SOLID – Single Responsibility Principle (SRP). Architektura, która przestrzega zasad SOLID cechuje się dużą stabilnością oprogramowania i elastycznym designem[11].

Jak wskazuje sama nazwa, metryka ta opisuje brak spójności, a więc pożądane jest aby była jak najmniejsza. Jest kilka sposobów jej obliczania.

1.2.2. Ocena efektywności pracy

Problem oceny efektywności pracy programisty nie jest problemem łatwym. Dostyć prosto można zaproponować podstawowe kryteria jednak nie zawsze oddają one faktyczny nakład pracy.

- czas pracy pracownika,
- ilość rozwiązanych zadań,

1.3. Proponowane rozwiązanie

- efektywność mierzona stosunkiem predyktywności do wykonanych zadań,
- ocena wygenerowanego kodu:
 - ilość wygenerowanego kodu,
 - rozkład kodu w plikach,
 - przejrzystość kodu,
 - jakość testów,
 - jakość dokumentacji,
 - nazewnictwo funkcji i klas,
 - analiza statyczna kodu,

1.3. Proponowane rozwiązanie

W niniejszej pracy utworzono wtyczkę do popularnego środowiska programistycznego Eclipse. Wtyczka pozwala na analizę rozkładu kodu w projekcie. Przegląda wszystkie pliki źródłowe w przestrzeni pracy, tworzy historię zmiany plików i pozwala na przejrzystą wizualizację zebranych danych.

1.3.1. Eclipse IDE

Eclipse [12] otwartoźródłową platformą na bazie której zostało stworzone zintegrowane środowisko programistyczne, głównie wykorzystywane do tworzenia projektów w języku Java, ale umożliwia wykorzystanie też innych popularnych języków programowania.

Samo środowisko Eclipse nie posiada narzędzi do tworzenia oprogramowania, a obsługuje wtyczki rozszerzające możliwości platformy. Tak więc Eclipse jest tylko programem obsługującym wtyczki i zapewniającym komunikację między nimi, przez co jest systemem łatwo rozszerzalnym. Implementacja środowiska w Javie zapewnia mu łatwą przenośność między systemami.

1.3.2. Eclipse PDE

Do środowiska Eclipse jest dostarczany plug-in PDE (*Plug-in Development Environment*) umożliwiający rozwijanie kolejnych wtyczek do programu Eclipse. Wtyczka zapewnia inną perspektywę (zmiana personalizacji okna głównego) przez co automatycznie po napisaniu kodu nowej wtyczki możemy przejść do jej kompilacji i debugowania. Minusem jest uruchamianie nowego okna systemu Eclipse, w którym uruchomiona jest wtyczka, co powoduje znaczne obciążenie pamięci operacyjnej komputera.

1.3.3. Zaimplementowane kryteria oceny kodu

W proponowanej wtyczce zostały zaimplementowane dwa podstawowe miary pracy programisty LOC i LOCPE. Zostały one przedstawione w sekcjach 1.2.1 i 1.2.1.

1.3.4. Utworzona wtyczka

Przechowywanie danych

Przechowywanie historii zmian plików dla danego projektu odbywa się poprzez zapisywanie pliku xml. Podczas uruchamiania wtyczki zostaje sprawdzony plik bazy danych projektu, jeśli nie istnieje zostaje utworzony nowy plik xml wraz z odczytem metryk dla stanu zastanego. Następnie z każdą wprowadzoną zmianą zostają zapisane zmiany dla plików które zostały zmienione z zapamiętanym stanem wcześniejszym dla danego pliku i projektu.

Struktura pliku xml zawierającego bazę danych dla wtyczki składa się z tagów *header*, *file* oraz *delta*. Node *file* odpowiada za opis śledzonych plików i ich stan aktualny. Każdy plik opisany jest atrybutem *path*, który zawiera pełną ścieżkę do pliku, i node'ami *loc* i *timestamp*, które określają odpowiednio ilość linii pliku i czas w jakim dany opis został zrobiony. Node *delta* przechowuje kolejne zmiany plików. Jest on określony poprzez atrybuty *changedFilesCount*, mówiący ile plików zostało zmienionych, *locpf* określający metrykę LOCPF w danej chwili, *timestamp* oznaczający czas zapisu i *trackedFilesCount*, który mówi ile plików było obserwowanych. W node *delta* występuje też node *file* określający pliki, których dotyczy *delta*.

Wszystkie zmiany w bazie danych następują po wykryciu zmiany w plikach przestrzeni roboczej, ale tylko plikach źródłowych przez co nie śledzone są pliki konfiguracyjne projektu, które ulegają zmianie z każdą operacją zapisu plików.

Widok i posługiwanie się wtyczką

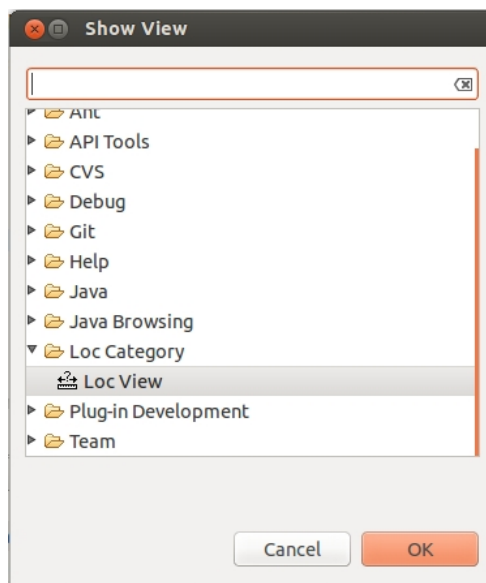
Po zainstalowaniu wtyczki pojawi się możliwość dodania nowego widoku, w którym wtyczka będzie widoczna. Aby wtyczka się uaktywniła należy przejść w górnym menu Eclipse *Window*→*Show View*→*Other...* w pojawiającym się okienku przechodzimy do *Loc category*→*Loc View*. Po tej operacji (pokazana na rysunku 1.1 wtyczka pokaże się na dolnym panelu wtyczek.

Po uruchomieniu wtyczki będzie dostępny podstawowy widok historii plików w projektach. Na dole okna uporządkowane są wszystkie projekty. Dla każdej zakładki wyświetlone są wszystkie pliki należące do danego projektu wraz historią 5 ostatnich zmian w ilości linii kodu (miara LOC). Jeżeli dla danego pliku nie istnieje zapis dla którejś ze zmian wtedy zostaje wyświetlony znak x. Widok pojedynczego projektu został pokazany na rysunku 1.2. Można też zwinąć pliki należące do jednego z projektów, wtedy będzie wyświetlana tylko nazwa danego projektu.

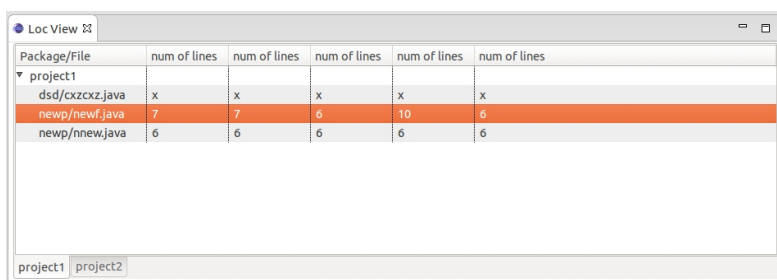
Dostęp do pozostałej historii zmian plików dostępny jest poprzez dwukrotne kliknięcie na nazwę pliku w widoku podstawowym. Operacja ta powoduje wyświetlenie okna dialogowego z wykresami. Domyślnie na pierwszym planie zostaje wyświetlony wykres ostatnich 100 zmian danego pliku. Jeśli plik posiada mniejszą bazę zmian zostaną wyświetlone wszystkie dostępne dane. Widok zakładki LOC przedstawiony jest na rysunku 1.3.

Na drugiej zakładce zamieszczony jest wykres drugiej miary, mianowicie LOCPF. Ilość wyświetlanych danych jest analogiczna do wykresu wskaźnika LOC. Widok drugiego wykresu został zamieszczony na rysunku 1.4.

1.3. Proponowane rozwiązanie

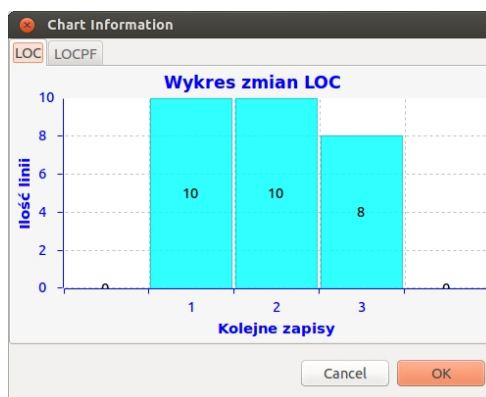


Rys. 1.1: Widok na wybór widoku wtyczki.



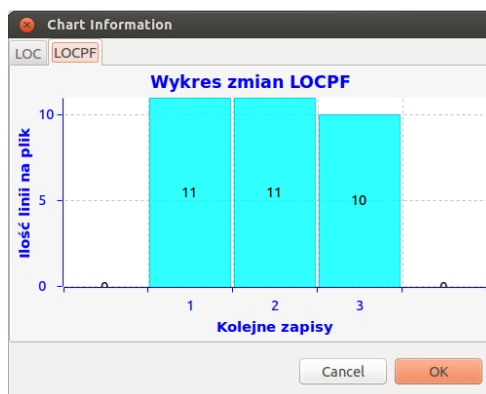
Package/File	num of lines	num of lines	num of lines	num of lines	num of lines
project1					
dsd/czxcz.java	x	x	x	x	x
newp/newf.java	7	7	6	10	6
newp/nnew.java	6	6	6	6	6

Rys. 1.2: Widok na podstawowy wygląd wtyczki.



Rys. 1.3: Widok na okno wykresów LOC.

1. Audyt w systemach informatycznych



Rys. 1.4: Widok na okno wykresów LOCPF.

Literatura

- [1] Software Freedom Conservancy. Git –everything is local. <http://git-scm.com/>, 2014.
- [2] The Apache Software Foundation. Apache subversion. <http://subversion.apache.org/>, 2014.
- [3] punct. Work easier work faster. <http://mercurial.selenic.com/>, 2005.
- [4] Jean-Philippe Lang. Overview redmine. <http://www.redmine.org/>, 2014.
- [5] Atlassian. Issue and project tracking software atlassian. <https://www.atlassian.com/software/jira>, 2014.
- [6] GitHub Inc. Features github. <https://github.com/features>, 2014.
- [7] Klocwork Inc. Source code analysis tools for software security and reliability klocwork. <http://www.klocwork.com/>, 2013.
- [8] LogiLab. Pylint - code analysis for python. <http://www.pylint.org/>, 2014.
- [9] GrammaTech Inc. Codesonar static analysis. <http://www.grammatech.com/codesonar>, 2014.
- [10] Jon McCormack, Damian Conway. Software characteristics and metrics. http://www.csse.monash.edu.au/~jonmc/CSE2305/Topics/07.14.SWEng2/html/text.html#what_is_a_metric, 2005.
- [11] Robert Cecil Martin. *Agile Software Development, Principles, Patterns, and Practices*. Prentice Hall, 2002.
- [12] The Eclipse Foundation. The eclipse foundation opensource community website. <http://www.eclipse.org/>, 2014.