Applying Machine Learning Techniques in Software Engineering

BITS ZG628T: Dissertation

by

Vijayshinva B. Karnure

2013HT13433

Dissertation work carried out at

HARMAN International (India) Pvt. Ltd. Bangalore

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE

PILANI (RAJASTHAN)

October 2015

Applying Machine Learning Techniques in Software Engineering

BITS ZG628T: Dissertation

by

Vijayshinva B. Karnure

2013HT13433

Dissertation work carried out at

HARMAN International (India) Pvt. Ltd. Bangalore

Submitted in partial fulfillment of M.Tech. Software Systems

Under the Supervision of

Raghuraman Rajagopalan, Director – Technology

HARMAN International (India) Pvt. Ltd. Bangalore

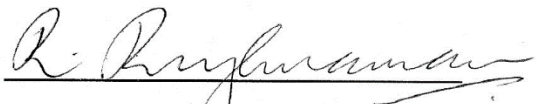
BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE

PILANI (RAJASTHAN)

October, 2015

CERTIFICATE

This is to certify that the Dissertation entitled **Applying Machine Learning Techniques in Software Engineering** and submitted by **Vijayshinva B. Karnure** having ID-No. **2013HT13433** for the partial fulfillment of the requirements of M.Tech. Software Systems degree of BITS, embodies the bonafide work done by him under my supervision.



Signature of the Supervisor

Place: BANGALORE

Date: 26 OCT 2015

Raghuraman Rajagopalan
Director – Technology
HARMAN International (India) Pvt. Ltd.
Bangalore

Birla Institute of Technology & Science, Pilani

Work-Integrated Learning Programmes Division

First Semester 2015-2016

BITS ZG628T: Dissertation

ABSTRACT

BITS ID No. : **2013HT13433**

NAME OF THE STUDENT : **Vijayshinva B. Karnure**

EMAIL ADDRESS : **2013ht13433@wilp.bits-pilani.ac.in**

STUDENT'S EMPLOYING ORGANIZATION & LOCATION : **HARMAN International (India) Pvt. Ltd. Bangalore**

SUPERVISOR'S NAME : **Raghuraman Rajagopalan**

SUPERVISOR'S EMPLOYING ORGANIZATION & LOCATION : **HARMAN International (India) Pvt. Ltd. Bangalore**

SUPERVISOR'S EMAIL ADDRESS: **Raghuraman.Rajagopalan@Harman.com**

DISSERTATION TITLE : **Applying Machine Learning Techniques in Software Engineering**

ABSTRACT

Most large scale software development projects follow an incremental build model where developers iteratively design, test and implement features. In an Agile environment requirements are broken down into features and a measure of complexity like story points is assigned to them. Developers are then assigned these development tasks based on their complexity. A software project hence can be considered as a group of software features with varied complexity, implemented by various developers.

Project Managers do not have a good way to compare these features in terms of complexity. It is usually based on the project manager's intuition. A tangible way, to gauge the effort put in by a developer to implement a feature, is the complexity of the code change set.

This is an attempt to use Machine Learning to categorize software feature implementations based on code change sets. The goal here is to cluster feature implementations, by extracting metrics from the code change sets of each implemented feature using a clustering algorithm. This clustering gives the Project Manager a tool to compare in-progress features with historical ones. The Project Manager can adjust timelines, risk categorization, testing strategies based on this.

Complexity comparision is crucial in software engineering, for tasks like determining developer productivity and better project planning.

Broad Academic Area of Work: **Machine Learning and Software Engineering**

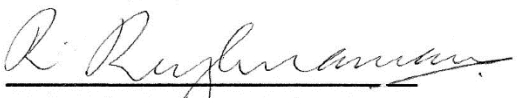
Key words: **Clustering, K-Means, Code Change Sets, Code Analysis**

Signature of the Student

Name: Vijayshinva B. Karnure

Date: 26/10/2015

Place: BANGALORE



Signature of the Supervisor

Name: Raghuraman Rajagopalan

Date: 26 OCT 2015

Place: BANGALORE

ACKNOWLEDGEMENTS

This project would not have been possible without the support of many people. I would like to acknowledge and extend my heartfelt gratitude to the following persons who have made the completion of this project possible.

My supervisor, Raghuraman Rajagopalan, for his full support and guidance throughout this project.

My additional examiner, Vijaya Kumar Reddy Gajjala for encouraging and supporting me to pursue the project.

Prof. S.R.K. Prasad Talasila from BITS Goa, for giving his feedback at various stages of the project which acted as a motivation for working on the project.

My colleague, Sagar Mattoo for proof-reading the dissertation thesis.

Last but not the least, I would like to express my love and gratitude to my beloved family, for their understanding & motivation, through the duration of this project.

Table of Contents

List of Figures

List of Abbreviations Used

API	Application Programming Interface
DVCS	Distributed Version Control System
PR	Pull Request
REST	Representational State Transfer
SaaS	Software as a service
Sklearn	SciKit-Learn Python Package
VCS	Version Control System

Chapter 1: Introduction

1.1 Current state of software development

Software development has come a long way since the seventies. Over the years several methodologies where developed and used. Now a days enterprises developing software have mostly settled into using an iterative incremental development model. Teams develop themes based on their project requirements. Themes are then broken down into self-contained units of work which are agreed upon by the developers and stakeholders. Developers own these units and work on them independently. Periodic integrations of these units happen as and when developers complete their implementation.

Open source projects rely on a community of developers who voluntarily contribute code to the project. Many successful community-led open source projects have been implemented over the years and the numbers continue to grow. In an open source project, developers contribute by submitting code changes that may implement new features or improve existing ones following a process agreed upon by the owners of the project.

When it comes to managing these software development projects, software development tools play an instrumental part. Any large scale software development project, either enterprise or open source at the bare minimum will need an Issue Tracking System and a Version Control System. The complexity rating or weightage given for each feature or module implementation is usually based on the project manager and developers intuition. Experience and familiarity of the project environment are major influences in deciding these scores.

The Issue Tracking System helps the team track and plan, tasks and issues. This is vital for proper collaboration between the team members and also multiple teams. Developers and stake holders can discuss, plan and prioritize their tasks. The Version Control System acts as the repository, for the code changes the team of developers is doing. The Version Control System tracks in detail, the changes to the code along with other metadata like the author of the change. The Version Control System and the Issue Tracking System are often integrated. This integration allows teams to map business requirements to feature implementations and ultimately the code change sets. The team can get a clear picture of every code change that was ever made, the reason it was made and references to the features added.

The Version Control System is a central piece in the entire puzzle of software development, as it is a record of the entire history of the project. Each feature or module implemented in a project has a corresponding code change set in the Version Control System. Current Version Control Systems do a good job of recording even minute details like the exact lines of code that were changed as part of a feature implementation.

The entire software project can be described as an amalgamation of these individual code change sets. The code change set is the tangible effort that was put in by the developer to implement a particular feature, module or fix. Various metrics can be extracted out of these code change sets. An analysis of these metrics can provide useful insights into the various features that were implemented for the project.

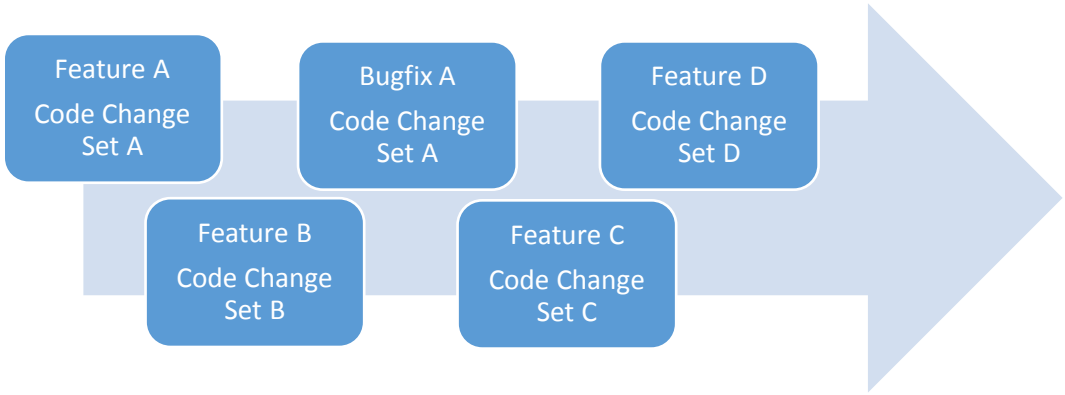


Figure 1: Software is an amalgamation of code change sets

As code change sets are the ultimate truth of a feature implementation in a software project, in this dissertation the metrics extracted from these code change sets are going to be used to relatively compare software features that were implemented. This relative comparison can help project manager to gauge their estimation in hindsight.

Chapter 2: Git Version Control System

2.1 The Distributed Version Control System

The Git Version Control System (Chacon & Straub, 2014) introduced in the year 2005, has grown leaps and bounds and is now a dominant player in the Version Control market. Most Version Control Systems prior to Git, were centralized. A central server maintained the source, along with its change history. Developers could check out files to make changes and implement new features.

Git changed the game by introducing a concept of Distributed Version Control System. Each client of the DVCs would basically mirror the entire repository from the central server. This way, the developers work independently on their own copy of the source code. When they are ready to contribute their changes, developers can merge their code changes with their peer repositories. In a DVCS all the clients basically act as backups for the repository, as each one has an exact clone of it.

Being a DVCS, Git does not require a central server. But a central Git server helps in collaboration. It acts as a starting point, from where new developers can clone their repositories. It also acts as a central repository to which developers can push their code changes and merge as required. The central server that maintains the repositories for all projects is called a Remote Git Repository.

SaaS providers like GitHub and BitBucket, provide Git **as a service** for clients to consume. Remote repositories can be hosted on GitHub or BitBucket at a nominal cost and the service providers take care of managing and administering the servers. GitHub boasts hosting over 28 million projects (About GitHub, 2015) as on date. These service providers also expose REST APIs which can be queried to get details about a Remote Git Repository.

2.2 Git Internals

A Git Repository is basically a folder structure which Git tracks. As and when developers make changes to this folder like adding new files, modifying existing files or deleting files, Git keeps track of all the changes. Periodically developers commit their changes. A commit creates a differential snapshot of the folder structure. A commit can act as a rollback point in case the developers want to revert their changes. Since these commits are differential snapshots Git also records the parent commit as part of the commit history.

By default every Git Repository has a default branch called the **master** branch. This branch is usually the ready to deploy branch, which gets deployed to production. Developers can create other branches from this main branch to work independently on their changes. With newer commits, branches move forward independently of each other. Branches can also be merged, where changes from one branch are pulled into another.

When developers are ready to contribute their changes to the main branch, an approval process is usually in place. The approval can be a peer review of the code or a simple sign off from the owner. This is achieved using a pull request in Git. A pull request encapsulates all the changes that were made to implement a particular feature. It can also be looked at as a series of code commits that the developer made as he/she implemented the feature.

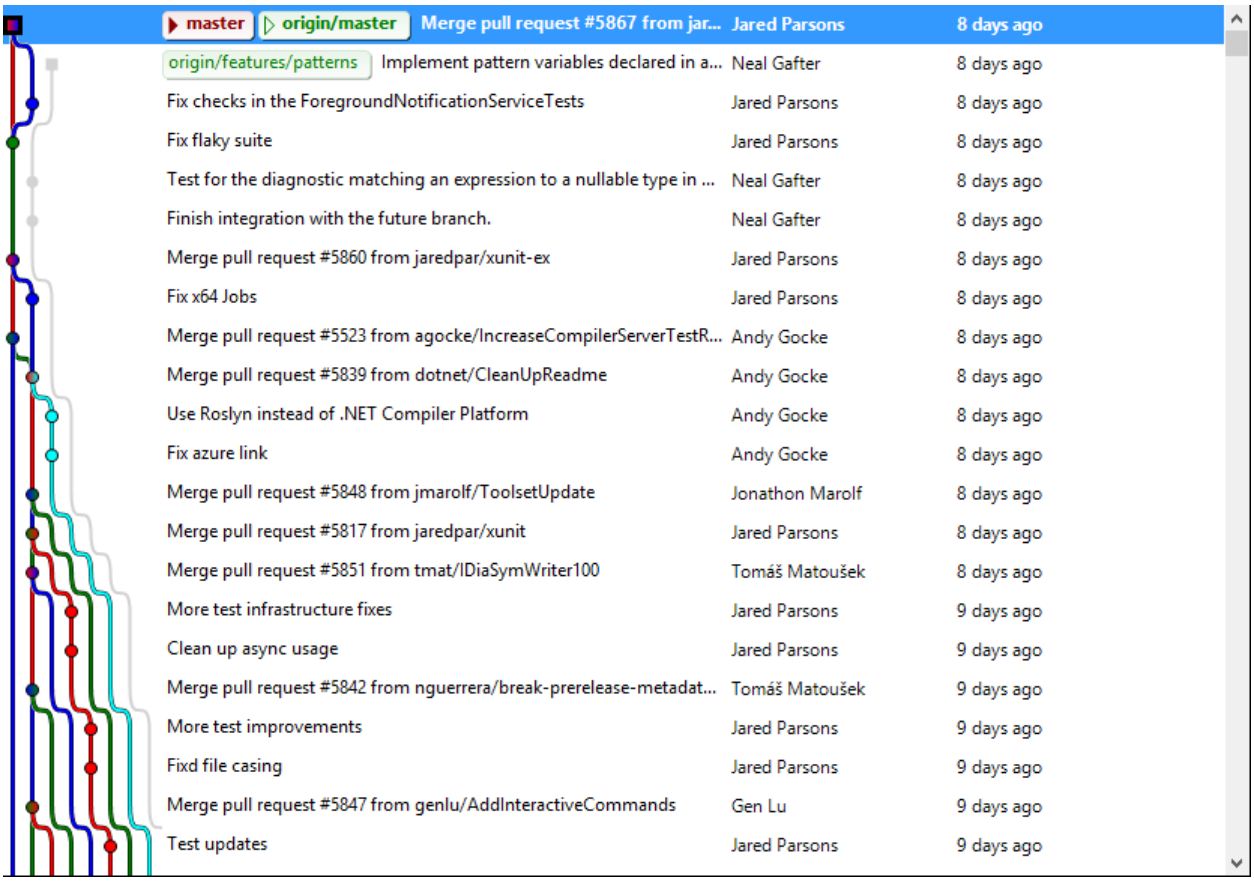


Figure 2: A pictorial representation of Git branching generated using the Git Extensions Tool for the GitHub project dotnet/roslyn

2.3 Developer Workflow

The industry is eagerly adopting Continuous Integration and Continuous Deployment. The project team is required to always have their code base **ready to deploy**. A typical developer workflow using Git is as follows.

1. A central project repository is created on a Git server.
2. This repository has a default source code branch called the master branch, which has the latest and greatest ready to deploy code for the project.
3. Developers clone this repository to start working on their changes
4. To implement a feature, developers create a branch out of the main branch.
5. Developers make changes to their local copy of the source code and commit changes on their respective branches.

6. Once the developers are convinced that the changes are good enough to be merged with the master branch, they create a pull request.
7. The pull request encapsulates changes to all the files that were made by a developer to implement a particular feature.
8. The pull request is then reviewed by the project owner, who either approves or rejects the changes.
9. Once the pull request is approved, the branch is merged into the master branch and is ready for deployment.

As and when developers contribute source code, the master branch keeps moving forward. Developers working on their respective branches periodically pull the master branch and merge it, so that they keep getting the latest updates. Project teams develop their own branching strategy based on their experiences. For example, some teams maintain a **futures** branch, where all new features are merged instead of the master branch. This keeps the master branch always ready to deploy and any new feature implementations do not interfere with the existing source code.

One way of looking at the entire project is as a series of pull requests merged together. Each pull request either added a new feature or fixed an existing issue. The code change that went in with each pull request is the tangible effort put in by the developer to implement the feature.

2.6 Metrics extracted from a Git Repository

Git exposes its internals using a standard API. The **libgit2** library is a native C library implementation that exposes all the functionality of Git. A .NET wrapper around this library is called the **libgit2sharp**. Similarly Git hosting services like Bitbucket and GitHub expose REST APIs, which can be queried to get details of Remote Git Repositories.

In this project GitHub REST APIs are used to get a list of all pull requests for a particular Git repository. Each pull request usually corresponds to a branch, a developer created to implement a feature and has a bunch of commits associated with it. Git APIs are then used to extract the code changes as part of these commits.

The following metrics are extracted for each pull request,

1. Count of files changed
2. Count of code lines added
3. Count of code lines removed

Apart from the above mentioned metrics, each line of code that was actually changed is also extracted for further analysis.

Chapter 3: Code Analysis

3.1 Code Metrics

Code metrics, were developed to answer the existential question of, "How complex is the code?" There are many well defined software metrics in use, in the industry today. Cyclomatic complexity, Maintainability index, Lines of code, Code coverage are a few to mention. Each metric, in its own way tries to standardize and quantize a measure that can define the characteristics of the software code.

In this study, the following metrics are extracted from code files,

1. Number of Expression Statements
2. Number of Branching Statements
3. Number of Looping Constructs
4. Number of Exception Handling Statements

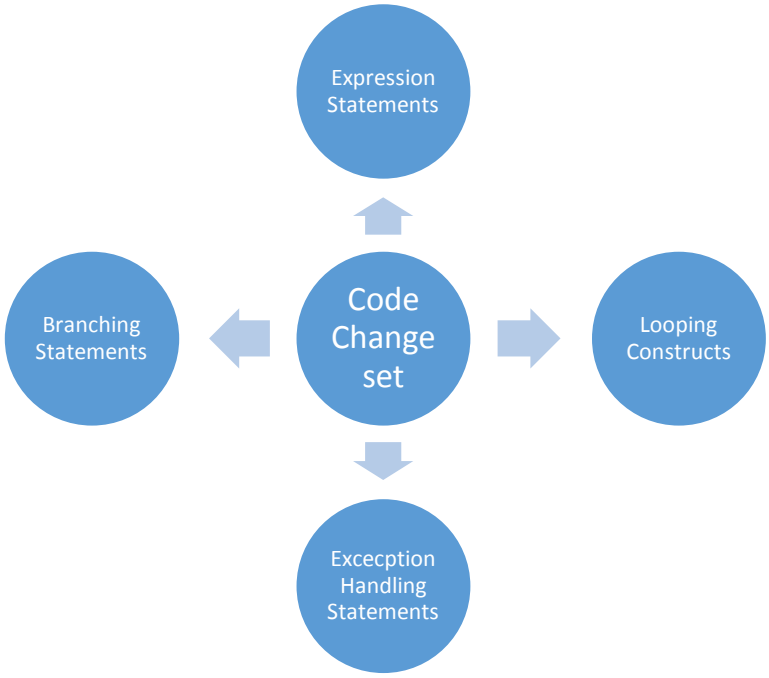


Figure 3: Metrics extracted from code files.

These metrics give us an idea of the composition of the code change set, in terms of the kinds of programming statements used. For example, if a particular pull request has more number of "If" statements we can conclude that the feature implemented, required more decision making.

3.2 Microsoft .NET Compiler Platform aka Project Roslyn

The Microsoft .NET framework is a popular development platform in the industry today. Its capabilities and the maturity of the tooling around it, has made it a popular choice for software implementation. An interesting project by Microsoft, codenamed Roslyn was to create an open source version of the .NET language

compilers. As part of the compiler project, code analysis was also built in. The .NET compiler platform exposes this code analysis functionality as a standard API.

Given a line of C# or VB.NET code, the Roslyn code analysis framework can categorize the statement based on its kind, like local declaration statement, if statement, expression statement etc.

Passing the lines of code extracted for each pull request, the Roslyn code analysis framework can churn out the count of each kind of language statement that was used to implement the feature.

The following metrics are extracted using the Roslyn code analysis framework,

1. Number Of Expression Statements
    a. Expression Statement
    b. Labeled Statement
    c. Using Statement
    d. Lock Statement
    e. Local Declaration Statement
    f. Empty Statement
    g. Unsafe Statement
    h. Fixed Statement
    i. Unchecked Statement
    j. Checked Statement
    k. Block Statement
2. Number of Branching Statements
    a. If Statement
    b. Continue Statement
    c. Return Statement
    d. Switch Statement
    e. Goto Statement
    f. GotoDefault Statement
    g. GotoCase Statement
3. Number of Looping Constructs
    a. For Statement
    b. Do Statement
    c. ForEach Statement
    d. While Statement
    e. YieldReturn Statement
    f. YieldBreak Statement
    g. Break Statement
4. Number of Exception Handling Statements
    a. Try Statement
    b. Throw Statement

These metrics are only calculated, if the file changed is written in C# or VB.NET. Other files like configuration files, project files are basically ignored.

These metrics roughly describe the composition of a code change set and can be used as a representation of a particular software feature implemented. A counter argument can be made, that different developers have different styles of coding and the same functionality can be written in different lines of code. That is true to some extent, but in most projects as developers are subjected to repeated code reviews, a consistency evolves. Automated code refactoring tools are also available and widely used by enterprises.

Chapter 4: Clustering the pull requests

4.1 Machine Learning using Python

Machine Learning concepts can be implemented using a wide variety of tools. SciKit-Learn (Pedregosa, et al., 2011) is a robust open source python library that can be used to implement machine learning concepts. SciKit-Learn provides most of the machine learning algorithms for classification, clustering and regression out of the box. Python distributions like Anaconda (Anaconda - Modern open source analytics platform, 2015) package together, all necessary python packages for data analysis making it easy for the community to install and use them.

4.2 K-Means Clustering

K-Means is a well-known unsupervised clustering algorithm which works with points in a vector space. K-Means groups together items in a dataset into K clusters usually based on their Euclidean distance measure. The items in a cluster will have lower Euclidean distance scores compared to ones outside the cluster.

K-Means works by

1. Choosing k centroids. The centroids can be random or use an initializing algorithm like kmeans++ (Arthur & Vassilvitskii, 2007) which initializes the centroid as distant from each other as possible
2. For each item in the data set calculate its distance from all centroids and assign it to the cluster of the nearest centroid.
3. Improve the centroids by computing the mean value of all the distances calculated in the previous step and choose them as the centroids.
4. Repeat step 2 and 3 until the centroids stop making any significant change in the clusters.

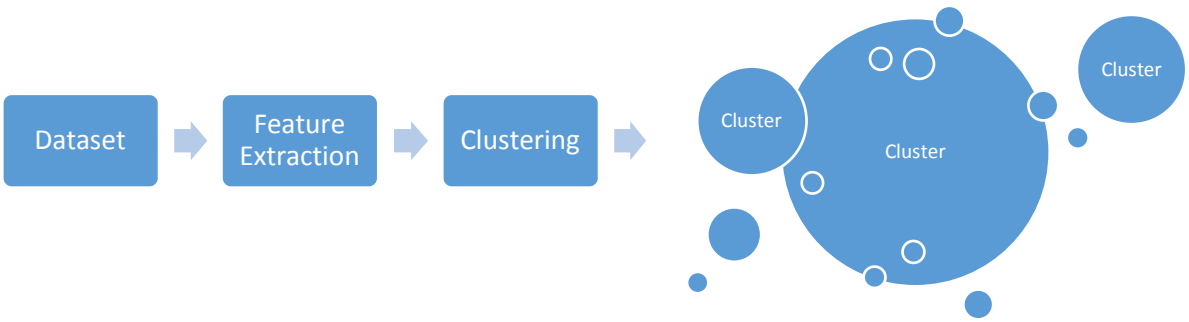


Figure 4: Machine Learning - Clustering

A good clustering run, should result in high inter cluster distances and low intra cluster distances. SciKit-Learn implements the KMeans algorithm for clustering data in the sklearn.cluster module. The KMeans algorithm is exposed as a class that implements a fit method, which learns the clusters from the input dataset. The **labels_** attribute holds the cluster labels that the algorithm generates.

The input dataset to the K-Means clustering algorithm, are the features derived from the merged pull requests. The dataset has seven dimensions namely

1. Count of files changed
2. Count of code lines added
3. Count of code lines removed
4. Number of Expression Statements
5. Number of Branching Statements
6. Number of Looping Constructs
7. Number of Exception Handling Statements

These seven dimensions roughly describe the composition of a change code set and can be used as a description of a particular software feature implementation.

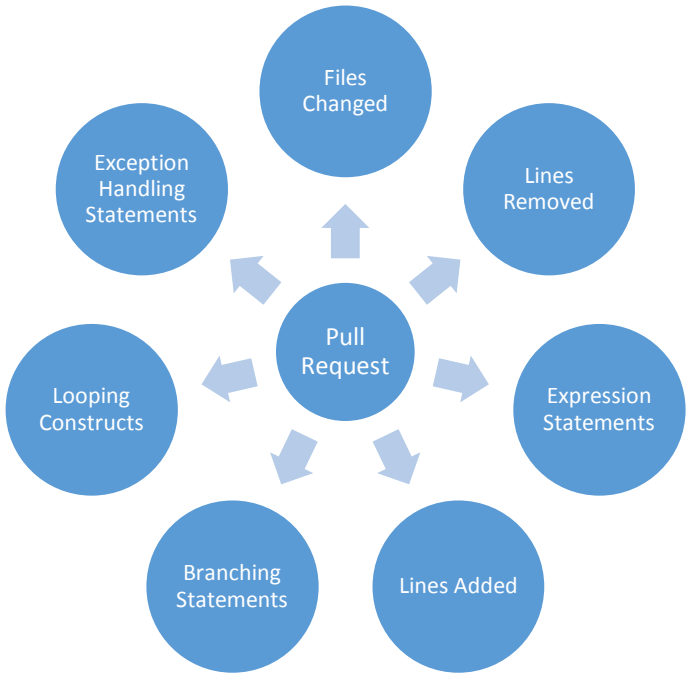


Figure 5: Features Extracted from Pull Requests

An issue with the KMeans algorithm is that it requires the number of clusters as an input. What is a good value for the number of clusters really depends on the input dataset. Silhouette analysis (Rousseeuw, 1987) can be used to determine the natural number of clusters for a dataset. The higher the separation between clusters, the higher is the value. A value of +1 indicates that the clusters are strongly separated. A value of -1 indicates that the clusters are vague. A value of 0 indicates a borderline case.

Silhouette analysis is susceptible to outliers in the dataset. If the dataset contains an outlier, a value of two for the number of clusters will show a high silhouette coefficient value. An acceptable target value for the silhouette coefficient value is 0.5, which gives good clusters in general, but further study is required.

Chapter 5: Putting it all together

5.1 Workflow

The following chart summarizes all the steps described in the earlier chapters.

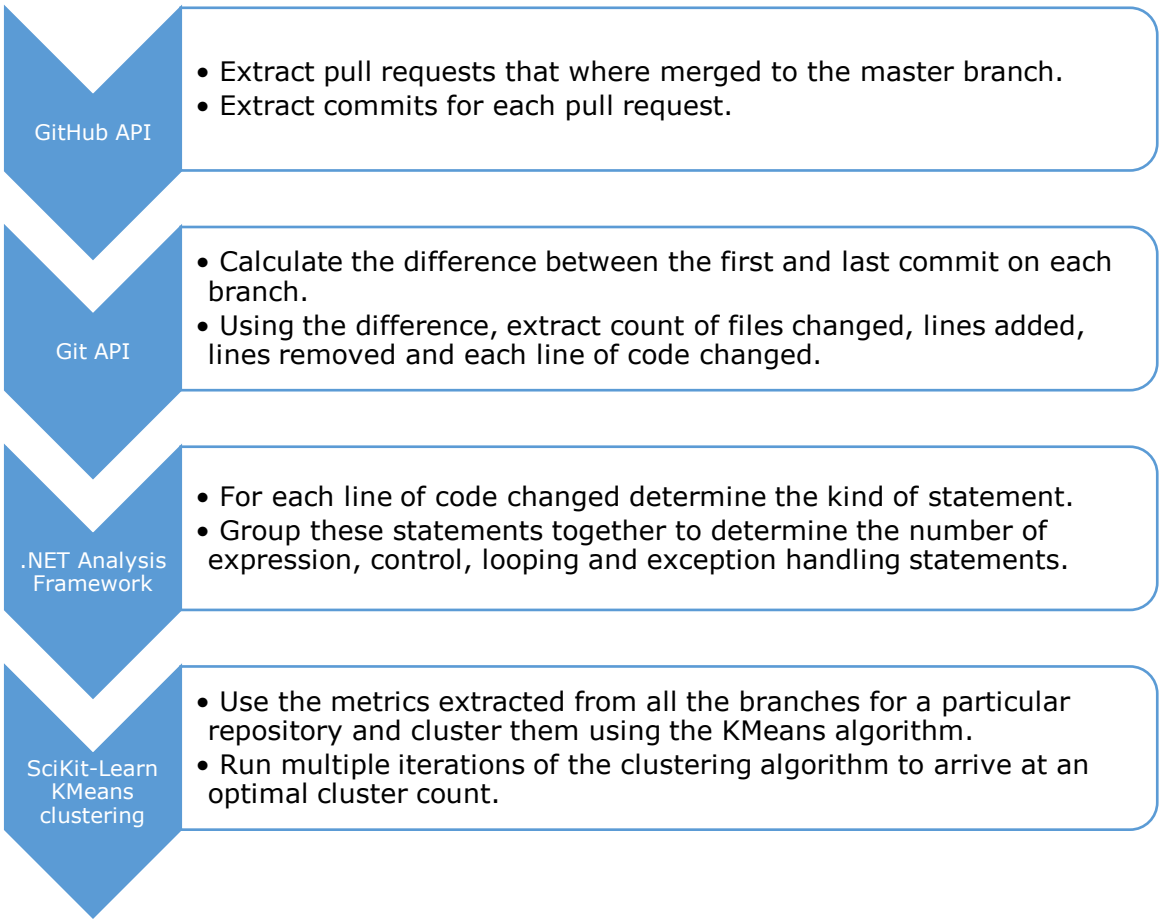


Figure 6: Workflow

This will result in clusters, where 'similar' pull requests end up together. For this, a tool called GitExtract was written in C# that performed the first three steps. Given a GitHub repository, GitExtract analyzes all the merged pull requests and extracts the seven defined metrics for each. The dataset generated is then fed to a Python script called clusterify.py which uses SciKit-Learn to generate clusters using the K-Means algorithm. The script runs the K-Means algorithm iteratively with varying K (number of clusters) and calculates the Silhouette scores. The number of clusters that achieves an approximate Silhouette score of 0.5 is used. The script then labels the input data set based on the clusters generated.

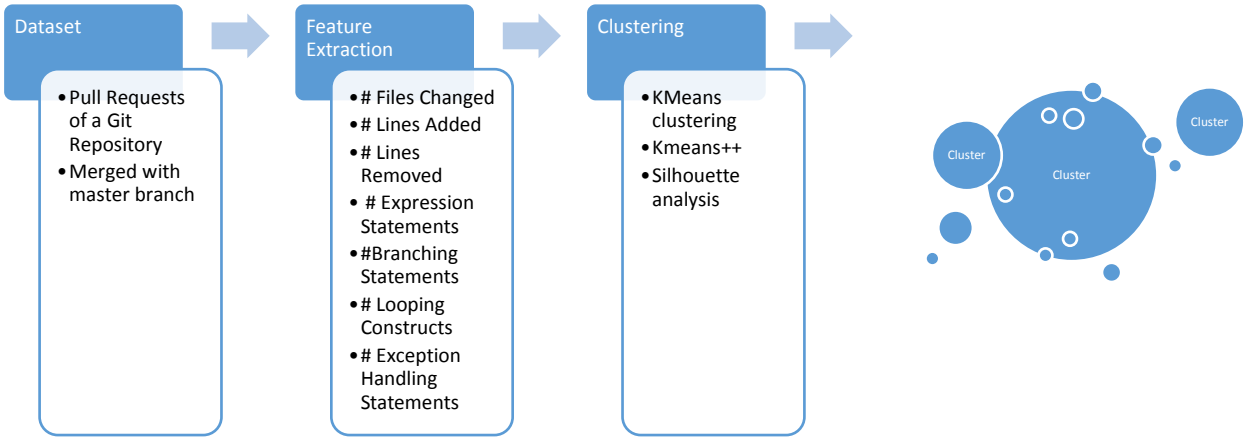


Figure 7: Machine Learning Workflow

5.2 A Case Study

The above approach was applied to an open source project repository hosted at https://github.com/dotnet/roslyn. This is the source code repository for the .NET compilers project and is continuously maintained and developed by the .NET team at Microsoft.

As on date 2254 pull requests where merged in the code repository. The run resulted in a [2254, 7] dataset, a sample of which is shown below.

	Branch	FileCount	LinesAdded	LinesDeleted	ExpressionStatements	BranchStatements	LoopStatements	ExceptionHandlingStatements
2	6169 Set State.WaitingForInput in P	4	47	39	9	-1	0	0
3	6158 Test `EmitSequenceOfBinaryEx	1	3	1	2	0	0	0
4	6147 Do not perform commit when '	2	49	0	49	0	0	0
5	6145 Temporarily disable linux vbc (	1	1	2	0	0	0	0
6	6144 Make dump directory predicta	1	22	2	16	1	1	2
7	6143 Add tests for use of unassigne	1	51	2	49	0	0	0
8	6130 Remove restore from targets f	1	1	32	0	0	0	0
9	6127 SynthesizedParameterSymbol	5	114	5	109	0	0	0
10	6120 Use the same .rsp file for VS RE	3	11	16	0	0	0	0
11	6118 Delete zip after use	1	3	0	0	0	0	0
12	6116 Update test resources propriet	60	662	216	0	0	0	0
13	6097 Rename MS.CA.Scripting.CSha	28	67	58	9	0	0	0
14	6092 Change several tests to WpfFa	1	5	5	0	0	0	0
15	6090 Remove left-over package.con	1	0	6	0	0	0	0
16	6089 Added supports clauses to pro	4	24	8	0	0	0	0
17	6086 Change AppConfigBasicFail to	1	4	3	1	0	0	0
18	6085 Porting fix for #4524 to stabiliz	3	214	2	212	0	0	0
19	6084 Do not crash on lexically bad fi	149	2180	185	37	2	0	4
20	6083 [AskMode] Change parsing of #	11	118	56	33	-1	0	0
21	6082 No longer call SetSynchronizat	3	63	18	40	5	0	-1
22	6080 Split the recommendation test	140	2097	151	0	0	0	0
23	6076 Allow goto in scripts [stabilizat	20	467	122	313	22	2	0

Figure 8: Dataset generated for dotnet/roslyn

105 clusters where created as it resulted in an approximate silhouette score of 0.5. A sample of the results are in Figure 9. The cluster labels in themselves do not mean anything, but from the results we can now infer that pull requests with the label 75 are 'similar'.

1	Pull Requests	Cluster Label
2115	687 Rename HasAccessChecksSuppressed to IgnoresAccessibil	74
2116	685 Check the CancellationToken in the DeclarePublicAPI anal	74
2117	684 Restore some access suppression tests in the binder need	74
2118	683 Prevent CodeActions commits during Inline Renames	74
2119	682 Add common Boxes to Microsoft.CodeAnalysis	75
2120	680 Fix a bug in AnalyzerManager.GetCompilationAnalysisSco	75
2121	677 Fix a NullReferenceException in the DeclarePublicAPI ana	75
2122	676 EnC: Calculate reverse map for each updated method cont	75
2123	673 Fix for issues around analyzer exception diagnostics gettir	75
2124	666 Handle OperationCanceledException in AnalyzerManager	75
2125	660 Handle TaskCancelledException in AnalyzerManager.GetA	75
2126	656 update Roslyn.Services.Editor.VisualBasic.UnitTests.dll ur	75
2127	655 Support EnC for lambdas & closures in VB compiler	75
2128	654 Format hash directives inside other hash directives of com	75
2129	653 Use ReportFatalError in TaskFactory helpers	76
2130	640 Add -u (unsign) option to FakeSign.exe	77
2131	639 Change SyntaxEditor.InsertAfter to call correct overload.	78
2132	638 Add documentation clarifying C# compiler behavior for sta	79
2133	627 add support for error list IsStable - progress indicator	79
2134	609 Improve SyntaxDiffer diffing rules	79
2135	606 Fix CompileAndVerifyOnWin8Only test utility	79

Figure 9: Resultant clusters of pull requests for dotnet/roslyn

Visualizing the resultant clusters is difficult as the input dataset has seven dimensions. To visualize the clusters in a 2D plot, the dataset is subjected to Principal Component Analysis (scikit-learn-developers, A demo of K-Means clustering on the handwritten digits data, 2015).

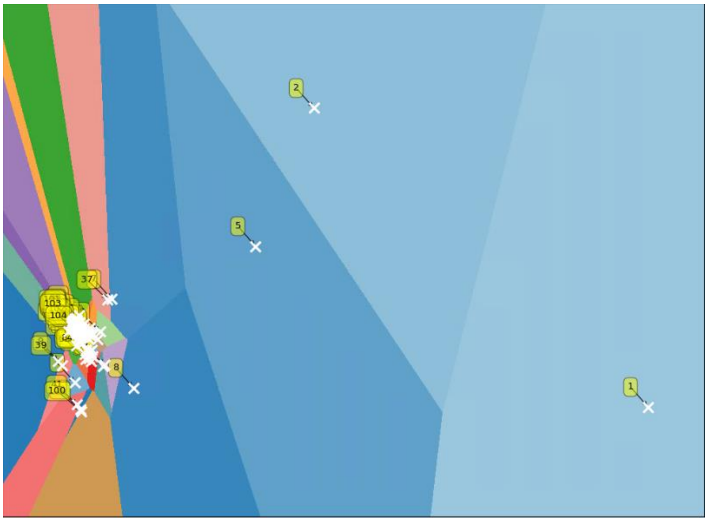


Figure 10: Visualization of the resultant clusters. Centroids of the clusters are marked.

Summary

This dissertation presents a study that uses Machine Learning techniques in Software Engineering. This study groups pull requests with similar code change sets together. The similarities are based on tangible metrics, generated from code change sets, implemented by developers. This grouping can help project managers gain useful insights into their projects.

In hindsight project managers can compare different pull requests. This can give project managers an insight into the effort put in by different developers. The model can be used for predictive analysis as well. Consider a software feature currently under development on an independent branch. Metrics derived from that branch can be used to figure out which cluster the branch currently lies in. This can help a project manager, take decisions based on experience of the earlier pull requests. The team can adjust timelines, risk categorization, testing strategies etc., based on other pull request experience.

Directions for future work

The work done in this dissertation can be further improved and expanded. Some of the ideas are listed below.

1. Outliers in the input dataset can skew the clustering. A mechanism to deal with outliers has to be implemented. Improvements in determining the number of clusters also needs to be worked on.
2. Infer additional data of pull requests from the linked Issue Management System like bug metrics, schedule adherence

References

About GitHub. (2015, October). Retrieved from GitHub: https://github.com/about

Anaconda - Modern open source analytics platform. (2015). Retrieved from Continuum Analytics: https://www.continuum.io/why-anaconda

Arthur, D., & Vassilvitskii, S. (2007). k-means++: The advantages of careful seeding. *Eighteenth annual ACM-SIAM symposium on Discrete algorithms.* Society for Industrial and Applied Mathematics.

Chacon, S., & Straub, B. (2014). *Pro Git.* Apress.

dotnet/roslyn. (2015). Retrieved from .NET Compiler Platform ("Roslyn"): https://github.com/dotnet/roslyn

GitHub API v3. (2015). Retrieved from GitHub Developer: https://developer.github.com/v3/

Hummel, J., & Neward, T. (2014, November). The Working Programmer : Rise of Roslyn. *MSDN Magazine*.

libgit2/libgit2sharp. (2015). Retrieved from LibGit2Sharp: https://github.com/libgit2/libgit2sharp

Neward, T., & Hummel, J. (2015, February). The Working Programmer - Rise of Roslyn, Part 2: Writing Diagnostics. *MSDN Magazine*.

Pedregosa, Varoquaux, Gramfort, Michel, Thirion, Grisel, . . . Duchesnay. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 2826-2830.

Rousseeuw, P. (1987). Silhouettes: a Graphical Aid to the Interpretation and Validation of Cluster Analysis. *Journal of Computational and Applied Mathematics*, 53-65.

scikit-learn-developers. (2015). *A demo of K-Means clustering on the handwritten digits data*. Retrieved from scikit-learn 0.16.1 documentation: http://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_digits.html

scikit-learn-developers. (2015). *A demo of K-Means clustering on the handwritten digits data*. Retrieved from scikit learn: http://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_digits.html

scikit-learn-developers. (2015). *Selecting the number of clusters with silhouette analysis on KMeans clustering*. Retrieved from scikit-learn 0.16.1 documentation: http://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html

scikit-learn-developers. (2015). *sklearn.cluster.KMeans*. Retrieved from scikit-learn 0.16.1 documentation: http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

Turner, A. (2014, Special connect() Issue). C# and Visual Basic - Use Roslyn to Write a Live Code Analyzer for Your API. *MSDN Magazine*.

Checklist of items for the Final Dissertation Report

1.	**Is the final report neatly formatted with all the elements required for a technical Report?**	Yes / ~~No~~
2.	Is the Cover page in proper format as given in Annexure A?	Yes / ~~No~~
3.	Is the Title page (Inner cover page) in proper format?	Yes / ~~No~~
4.	(a) Is the Certificate from the Supervisor in proper format? (b) Has it been signed by the Supervisor?	Yes / ~~No~~ Yes / ~~No~~
5.	Is the Abstract included in the report properly written within one page? Have the technical keywords been specified properly?	Yes / ~~No~~ Yes / ~~No~~
6.	Is the title of your report appropriate? **The title should be adequately descriptive, precise and must reflect scope of the actual work done.** Uncommon abbreviations / Acronyms should not be used in the title	Yes / ~~No~~
7.	Have you included the List of abbreviations / Acronyms?	Yes / ~~No~~
8.	Does the Report contain a summary of the literature survey?	Yes / ~~No~~
9.	Does the Table of Contents include page numbers? (i). Are the Pages numbered properly? (Ch. 1 should start on Page # 1) (ii). Are the Figures numbered properly? (Figure Numbers and Figure Titles should be at the bottom of the figures) (iii). Are the Tables numbered properly? (Table Numbers and Table Titles should be at the top of the tables) (iv). Are the Captions for the Figures and Tables proper? (v). Are the Appendices numbered properly? Are their titles appropriate	 Yes / ~~No~~ Yes / ~~No~~ Yes / ~~No~~ Yes / ~~No~~ Yes / ~~No~~
10.	Is the conclusion of the Report based on discussion of the work?	Yes / ~~No~~
11.	Are References or Bibliography given at the end of the Report? Have the References been cited properly inside the text of the Report? Are all the references cited in the body of the report	Yes / ~~No~~ Yes / ~~No~~ Yes / ~~No~~
12.	Is the report format and content according to the guidelines? The report should not be a mere printout of a Power Point Presentation, or a user manual. Source code of software need not be included in the report.	Yes / ~~No~~

Declaration by Student:

I certify that I have properly verified all the items in this checklist and ensure that the report is in proper format as specified in the course handout.

Place: BANGALORE

Date: 26/10/2015

Signature of the Student

Name: VIJAYSHINVA B. KARNURE

ID No.: 2013HT13433