Defect Prediction in Software Engineering

Daniel Rodriguez and Javier Dolado

2017-06-24

Contents

Welcome

Defect Prediction in Software Engineering. Work in progress.

Part I

Introduction to Defect Prediction in Software Engineering

Defect Prediction in Data Mining

Defect Classification

Defect Prediction and Ranking

Standards to deal with defects

ODC (Orthogonal defect classification)

AutoODC

Part II

Data Sources in Software Engineering
in Software Engineering

Chapter 1

Data Sources in Software Engineering

We classify this trail in the following categories:

- *Source code* can be studied to measure its properties, such as size or complexity.
- *Source Code Management Systems* (SCM) make it possible to store all the changes that the different source code files undergo during the project. Also, SCM systems allow for work to be done in parallel by different developers over the same source code tree. Every change recorded in the system is accompanied with meta-information (author, date, reason for the change, etc) that can be used for research purposes.
- *Issue or Bug tracking systems* (ITS). Bugs, defects and user requests are managed in ISTs, where users and developers can fill tickets with a description of a defect found, or a desired new functionality. All the changes to the ticket are recorded in the system, and most of the systems also record the comments and communications among all the users and developers implied in the task.
- *Messages* between developers and users. In the case of free/open source software, the projects are open to the world, and the messages are archived in the form of mailing lists and social networks which can also be mined for research purposes. There are also some other open message systems, such as IRC or forums.
- *Meta-data about the projects.* As well as the low level information of the software processes, we can also find meta-data about the software projects which can be useful for research. This meta-data may include intended-audience, programming language, domain of application, license (in the case of open source), etc.
- *Usage data.* There are statistics about software downloads, logs from servers, software reviews, etc.

1.1 Types of information stored in the repositories

- Meta-information about the project itself and the people that participated.
  - Low-level information
    - Mailing Lists (ML)
    - Bugs Tracking Systems (BTS) or Project Tracker System (PTS)
    - Software Configuration Management Systems (SCM)
  - Processed information. For example project management information about the effort estimation and cost of the project.
- Whether the repository is public or not

- Single project vs. multiprojects. Whether the repository contains information of a single project with multiples versions or multiples projects and/or versions.
- Type of content, open source or industrial projects
- Format in which the information is stored and formats or technologies for accessing the information:
  - Text. It can be just plain text, CSV (Comma Separated Values) files, Attribute-Relation File Format (ARFF) or its variants
  - Through databases. Downloading dumps of the database.
  - Remote access such as APIs of Web services or REST

Chapter 2

Repositories

There is a number of open research repositories in Software Engineering. Among them:

- FLOSSMole (?) http://flossmole.org/
- FLOSSMetrics (?): http://flossmetrics.org/
- PROMISE (PRedictOr Models In Software Engineering) [8]: http://openscience.us/repo/
- Qualitas Corpus (QC) (?): http://qualitascorpus.com/
- Sourcerer Project (?): http://sourcerer.ics.uci.edu/
- Ultimate Debian Database (UDD) (?) http://udd.debian.org/
- SourceForge Research Data Archive (SRDA) (?) http://zerlot.cse.nd.edu/
- SECOLD (Source code ECOsystem Linked Data): http://www.secold.org/
- Software-artifact Infrastructure Repository (SIR) [http://sir.unl.edu]
- OpenHub: https://www.openhub.net/

Not openly available:

- The International Software Benchmarking Standards Group (ISBSG) http://www.isbsg.org/
- TukuTuku http://www.metriq.biz/tukutuku/

Some papers and publications/theses that have been used in the literature:

- Helix Data Set (?): http://www.ict.swin.edu.au/research/projects/helix/
- Bug Prediction Dataset (BPD) (?, ?): http://bug.inf.usi.ch/
- Eclipse Bug Data (EBD) (?, ?): http://www.st.cs.uni-saarland.de/softevo/bug-data/eclipse/

2.1 Some Tools/Dashboards to extract data

Within the open source community, two toolkits allow us to extract data that can be used to explore projects:

Metrics Grimoire http://metricsgrimoire.github.io/

SonarQube http://www.sonarqube.org/

Chapter 3

Metrics in Soft Eng Prediction

3.1 Complexity Metrics

These metrics have been used for quality assurance during:

- Development to obtain quality measures, code reviews etc.
- Testing to focus and prioritize testing effort, improve efficiency etc.
- Maintenance as indicators of comprehensibility of the modules etc.

Generally, the developers or maintainers use rules of thumb or threshold values to keep modules, methods etc. within certain range. For example, if the cyclomatic complexity ($v(g)$) of a module is between 1 and 10, it is considered to have a very low risk of being defective; however, any value greater than 50 is considered to have an unmanageable complexity and risk. For the essential complexity ($ev(g)$), the threshold suggested is 4 etc. Although these metrics have been used for long time, there are no clear thresholds, for example, although McCabe suggests a threshold of 10 for $v(g)$, NASA's in–house studies for this metric concluded that a threshold of 20 can be a better predictor of a module being defective.

Several authors have studied the relation between lines of code and defects, for example, (**?**)

3.1.1 McCabe

(**?**)

(**?**)

3.1.1.0.1 Halstead y su ciencia del software.-

A finales de 1970, Halstead desarrolló un conjunto de métricas conocidas en su conjunto como la *ciencia del software de Halstead.* Estas métricas se basan en último término en computar los *operadores* y *operandos* de un programa:

- Los operadores son las palabras reservadas del lenguaje (tales como `if`, `while` o `for`), los operadores aritméticos (`+`, `-`, `*`, etc.), los de asignación (`=`, `+=`, `*=`, etc.) y los operadores lógicos (AND, OR, etc.)
- Los operandos son las variables, los literales y las constantes del programa.

Halstead propone diferentes medidas que basa en el cálculo previo del número de operadores y operandos únicos, y del número total de operadores y operandos. Para calcular todos estos valores utiliza la siguiente notación:

- n_1, number of distinct operators

Table 3.1: Attribute Definition Summary

	Metric	*Definition*
McCabe	*LoC*	McCabe's Lines of code
	$v(g)$	Cyclomatic complexity
	$ev(g)$	Essential complexity
	$iv(g)$	Design complexity
Halstead base	*uniqOp*	Unique operators, n_1
	uniqOpnd	Unique operands, n_2
	totalOp	Total operators, N_1
	totalOpnd	Total operands, N_2
Halstead Derived	n	Vocabulary, $n = n_1 + n_2$
	L	Program length, $N = N_1 + N_2$
	V	Volume, $V = N \cdot log_2(n)$
	d	Difficulty $D = 1/L$
	i	Intelligence
	e	Effort $e = V/L$
	b	Error Estimate
	t	Time $T = E/18$ seconds
	lOCode	Line count of statement
	lOComment	Count of lines of comments
	lOBlank	Count of blank lines
	lOCodeAndComment	Count of lines of code and comments
Branch	*branchCount*	No. branches of the flow graph
Class	true, false	Reported defects?

- N_1, total number of operators
- n_2, number of distinct operands
- N_2, the total number of operands

For example, given the following excerpt:

```
if (MAX < 2) {
    a = b * MAX;
    System.out.print(a);
}
```

We obtain the following values:

- $n1 = 6$ (if, { }, System.out.print(), =, *, <)
- $N1 = 6$ (if, { }, System.out.print(), =, *, <)
- $n2 = 4$ (MAX, a, b, 2)
- $N2 = 6$ (MAX, 2, a, b, MAX, a)

Using this four variable, several metrics are calculated.

{Vocabulario} ($n = n_1 + n_2$). El vocabulario es una medida de la complejidad de las sentencias de un programa a partir del número de operadores y operandos únicos. Se basa en el hecho de que un programa que utiliza un número reducido de elementos muchas veces será, según Halstead, menos complejo que un programa que emplea un mayor número de elementos.

{Longitud} ($N = N_1 + N_2$). La longitud es una medida del tamaño de un programa: cuanto más grande, mayor será la dificultad para comprenderlo. Se trata de una medida alternativa a la simple cuenta de líneas de código y casi igual de fácil de calcular. N es sin embargo más sensible a la complejidad, porque no asume que todas las instrucciones son igualmente fáciles o difíciles de entender. % La longitud puede ser estimada con la ecuación: % $\hat{N} = N_1 log_2 n_1 + N_2 log_2 n_2$.

{Volumen} ($V = N \cdot log_2(n)$). El vocabulario se define como el número de bits necesarios para codificar un programa en un alfabeto que utiliza un único carácter para representar todo operador u operando. Mientras