How to Measure and Estimate Software Maintainability for Open Source Projects?

Celia Chen

Agenda

- Systematic literature review on Maintenance Effort Estimation for Open Source Projects
- A comparison study on automated metrics and human-assessed metrics

Why Maintainability?

Low Maintainability

Difficult to modify

Increase the participation cost

Difficult to find solutions for bugs

Increase maintenance effort

Why Maintainability?

Low Maintainability

Di

A successful OSS project requires to be highly maintainable

Difficult to find solutions for bugs

Increase maintenance effort

Results from the Empirical Study on MI

- We performed an empirical analysis on Maintainability Index.
  - Among 97 OSS projects written in Java, PHP and Python, Maintainability Index differs across three languages at 90% confidence level.
  - Among 97 OSS projects in the domains of Web Development Framework, Audio & Video, Security, Testing Tools and System Administration, Maintainability Index differs across five domains at 95% confidence level.

Pros & Cons of Maintainability Index

Pros:

-Very popular and often used in maintenance practice
-Very easy to use

Cons:

-MI is a composite metrics and as such it is hard to determine which of the metrics cause a particular total value for MI
-The different metric values depend on the type of programming language, the programmer, the perception of the quality of code, etc.

How do others measure OSS maintainability?

- In order to understand its current state of the art and to identify opportunities for future research, we performed a systematic literature review on OSS maintenance effort estimation approaches.
- This paper was accepted and published in the ICSME 2016 proceedings.


Systematic Literature Review on Maintenance Effort Estimation for Open Source Projects
A comparison study on automated and human-assessed maintainability metrics

Research Questions

RQ1: What evidence is there for maintenance effort estimation techniques/methods for OSS projects?

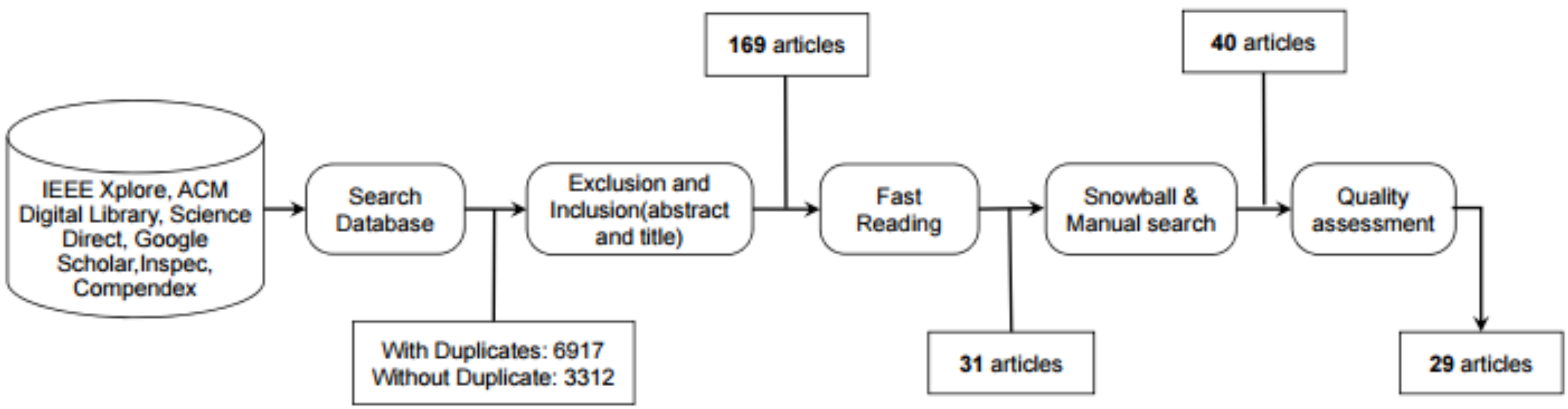
RQ2: What metrics related to OSS development records are extracted for maintenance effort estimation and how can they be classified?

RQ3: What are common projects and the size of dataset used as study cases in OSS maintenance effort estimation, and how has the frequency of approaches related to the size of dataset?

RQ4: What methods/approaches are used to estimate actual project maintenance effort (including those from the usual incomplete OSS development records)?

RQ5: What is the overall estimation accuracy of OSS maintenance effort estimation?

Study Selection



RQ1: Classification and Research Type

Topic	Sub-topic
Indirect effort prediction of the entire project	Source code-based estimation
	Process-based estimation
Direct effort prediction of the entire project	People-based estimation
	Activity-based estimation
Effort prediction of maintenance activity	Peer code review
	Duplicate issues identification
	Bug fixing
Individual contribution measurement	
Guidelines and discussion	

- Studies aim to predict effort of maintenance activity mainly concentrated on bug fixing time prediction.
- Less efforts contributed to other types of activities such as peer code review and duplicate issues identification.
- The highest frequency was in the research for developing estimation method. Within this method, the highest frequency was in predicting maintenance activity effort.

RQ2: Metrics/Factors

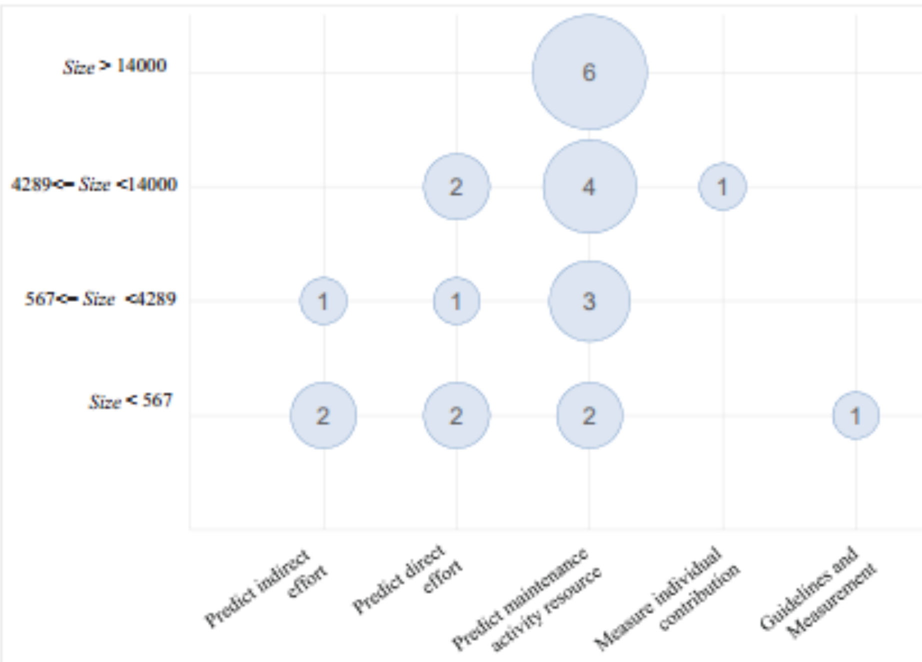
- There are 85 metrics in total.
- Priority of bug has the highest support with nine studies used it in their estimation models.

Topic	Type of Metric Set
Predict indirect effort of the entire project	Project{Size, Task}, Changes{CC, Function}
Predict direct effort of the entire project	Project{Time, Commits, Developer}, Participant{Bug Collaborator}, Community{Contributor}
Predict effort of maintenance activity	Project{Size, Time, Task, Bugs}, Changes{ALL}, Issue Report{ALL}, Participant{Bug Reporter, Bug Collaborator}, Community{workload}
Measure individual contribution	Participant{ALL}, Community{Activity}
Guidelines and discussion	Project{Size}, Changes{CC}

- Severity was also commonly used in estimation.

The detailed descriptions of these metrics can be found on: http://itechs.iscas.ac.cn/cn/membersHomepage/wuhong/metrics.html

RQ3: Studied Projects and Dataset



Most selected studies under the topic of predicting maintenance activity time used bigger datasets but most studies under the topic of predicting maintenance effort of entire projects both directly and indirectly used smaller datasets.

RQ4: Estimation Methods

The results show that estimation models for entire projects adopted a diversity of types of metrics while using linear model or classification methods.

The estimation models for maintenance activity are opposite.

The issue-report related metrics are the main metrics while these models adopted a diversity of methods.

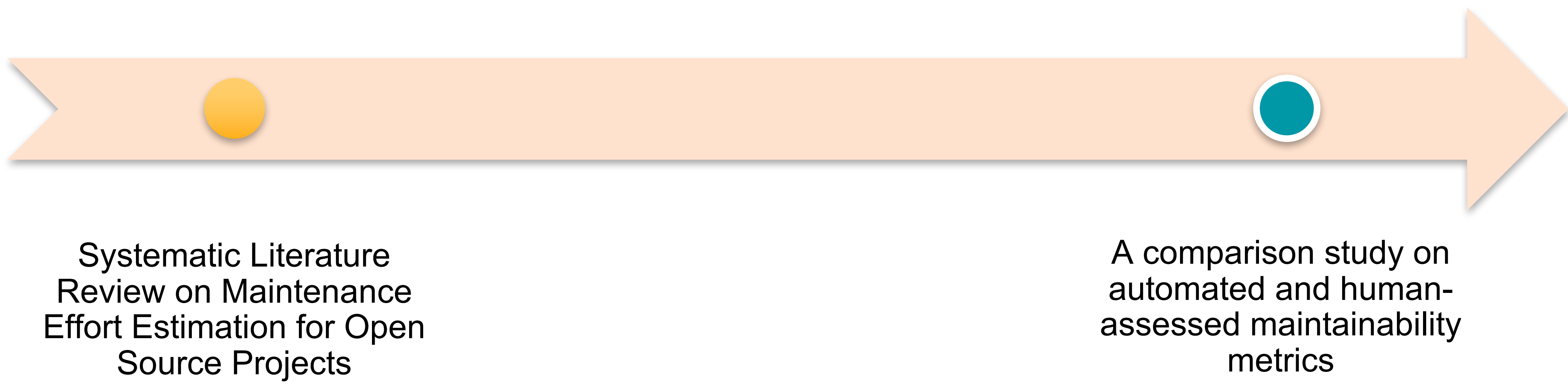
RQ5: Estimation Accuracy

20 out of the 29 selected paper presented some sort of evaluation methods, whether mathematical or descriptive.

Topic	Sub-topic	Estimation Method
Indirect effort prediction of the entire project	Source code based estimation	Linear regression model
	Processes-based estimation	Classification
Direct effort prediction of the entire project	People-based	Manpower function
		Classification
	Activity-based	Linear model
Effort prediction of maintenance activity	Effort for peer code review	Non-linear equation
	Effort for duplicateissues identification	Classification model
	Bug fixing time	linear regression
		Decision tree
		Support Vector Machine (SVM)
		K-Nearest Neighbor (KNN)
		Apriori & K-means
		Logistic Regression
		Naïve Bayes
		Distribution Functions
		Average weighted similarity

Discussions

- New evaluation methods are needed to validate the correctness of these estimation methods.
- Maintenance cost estimation models of OSS projects are different with general software system.
- Studies that can quantitatively infer OSS maintenance effort from size-related metrics are needed.
- It will be worthwhile to explore the capability model for OSS developers.


Systematic Literature Review on Maintenance Effort Estimation for Open Source Projects
A comparison study on automated and human-assessed maintainability metrics

Study Design

Goal: to compare the human-assessed maintainability metrics with the automated maintainability metrics counterparts.

Research Question: *Which metrics can more accurately reflect software maintainability?*

TABLE I
CHARACTERISTICS OF PROJECT DATA SOURCES

Language	Number of Projects	Average SLOC
Java	6	35,200
PHP	5	67,145

Context Selection:

- 11 open source projects found on Sourceforge and Apache

This study was accepted and published in NASAC 2017 and ICSE Poster 2017.

Automated Maintainability Metrics Assessment

TABLE II
AUTOMATED MAINTAINABILITY METRICS MEASURED IN THE CONTEXT OF RQ

Metrics	Description / Equation
Technical Debt (TD)	The total amount of effort in man-hours is required in order to reimburse all debts in the project
Lines of Code (LOC)	The number of lines of code (Excluding comments and white spaces)
Technical Debt Density (TDD)	TD per LOC
Maintainability Index without Comments (MIwtC)	An index value excluding code comments that represents the relative ease of maintaining the code.
Maintainability Index with Comments (MIwC)	An index value including code comments that represents the relative ease of maintaining the code.
Maintainability Index (MI)	Sum of MIwtC and MIwC
Halstead Volume (HV)	Halstead complexity measures
Code Complexity (CC)	McCabe's cyclomatic complexity
Comment Ratio (CR)	The percentage of comments

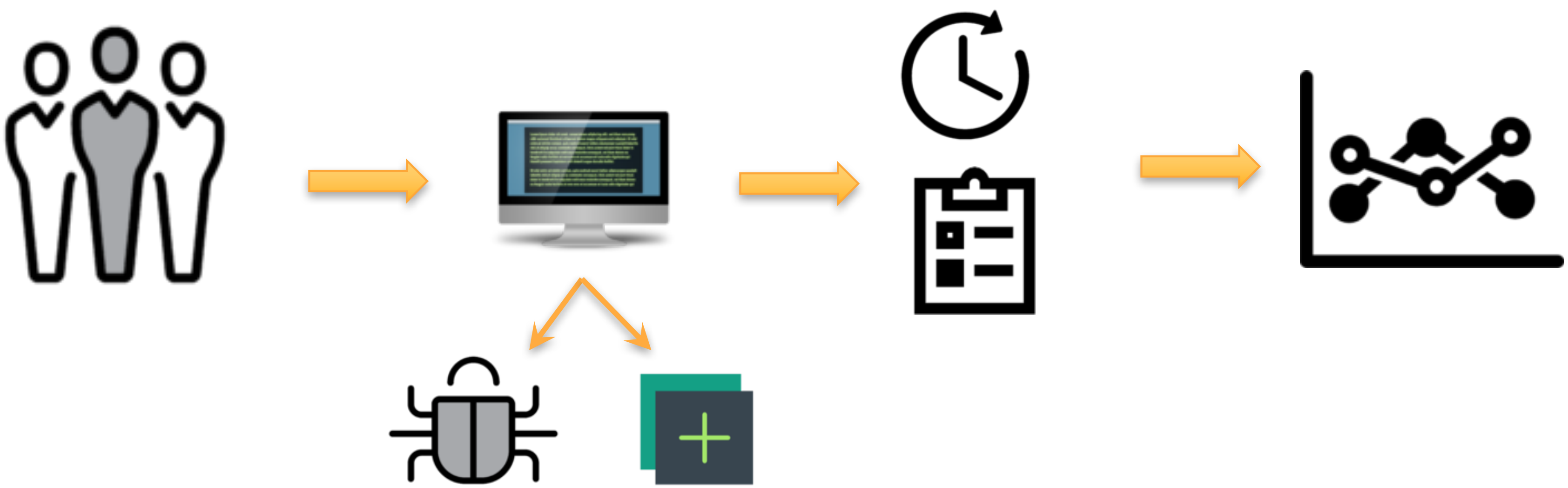
Human-assessed Maintainability Metrics Assessment

- Participants: six recruited developers
- Task: to perform maintenance tasks (bug fixing or new feature requests implementation) on 11 open source projects
- Collected metrics:
    - Developers: Overall industry experience, OSS experience
    - Tasks: Task difficulty, average time spent on task, task completion
    - COCOMO II SU factors: Factor rating and rationale

COCOMO II Software Understandability Factors

Factor	*Very Low*	*Low*	*Nominal*	*High*	*Very High*
Structure	Very low cohesion, high coupling, spaghetti code.	Moderately low cohesion, high coupling.	Reasonably well structured; some weak areas.	High cohesion, low coupling.	Strong modularity, information hiding in data/control structures.
Application Clarity	No Match between program an application worldviews.	Some correlation between program and application.	Moderate correlation between program and application.	Good correlation between program and application.	Clear match between program and application worldviews.
Self-Descriptiveness	Obscure code; documentation missing, obscure or obsolete.	Some code commentary and headers; some useful documentation.	Moderate level of code commentary, headers, documentation.	Good code commentary and headers; useful documentation; some weak areas.	Self-descriptive code; documentation up-todate, wellorganized, with design rationale.

Experiment Process



Analysis Results

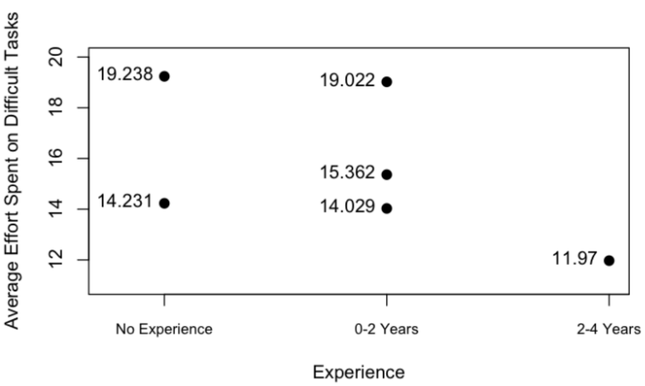
- Pearson product-moment correlation:
    - *Automated Maintainability Metrics Result:*
        - MI, MIwC, and CR showed strong negative correlation with actual effort spent
        - TD, TDD, MIwtC, HV, CC, and LOC are not correlated
    - *Human-assessed Maintainability Metrics Result:*
        - Structure, Application Clarity and Self-Descriptiveness are strongly correlated
        - Documentation quality is not correlated

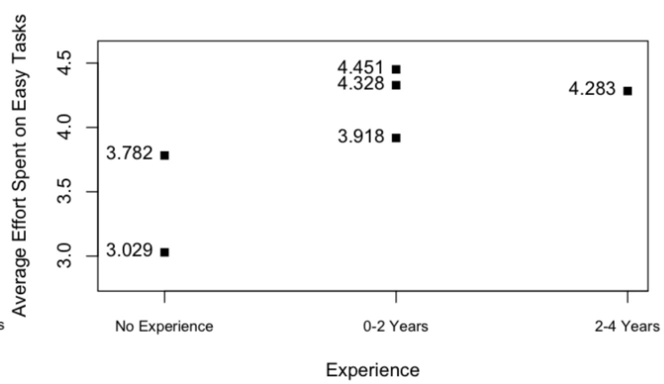
Developers Experience

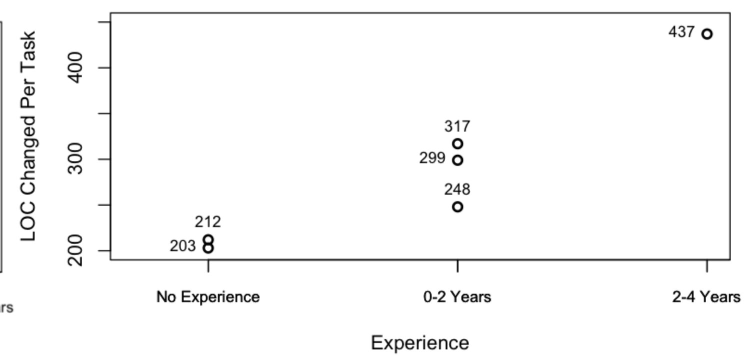
- Developers with more experience complete higher percentage of tasks.

- Developers with more experience spend less effort on hard tasks. However, the differences on easy tasks are very minor.

- Developers with more experience change more lines of existing code when performing maintenance tasks.







Comparison Results and its Application

- The results suggest that **human-assessed maintainability metrics** may be a better and more accurate alternative to estimate software maintainability.
- However, in practice, the automated metrics such as TD and MI approaches can efficiently help prioritize the parts of the software that need the most attention.
- Both human assessed and automated approaches can be synergetic.

Conclusions

- Maintainability Index and Technical Debt can be used as an indicator for software maintainability but with limitations.
- Automated maintainability approaches need to be combined with human-assessed approaches to better measure software maintainability.