SYSTEMS ENGINEERING RESEARCH CENTER

SMU

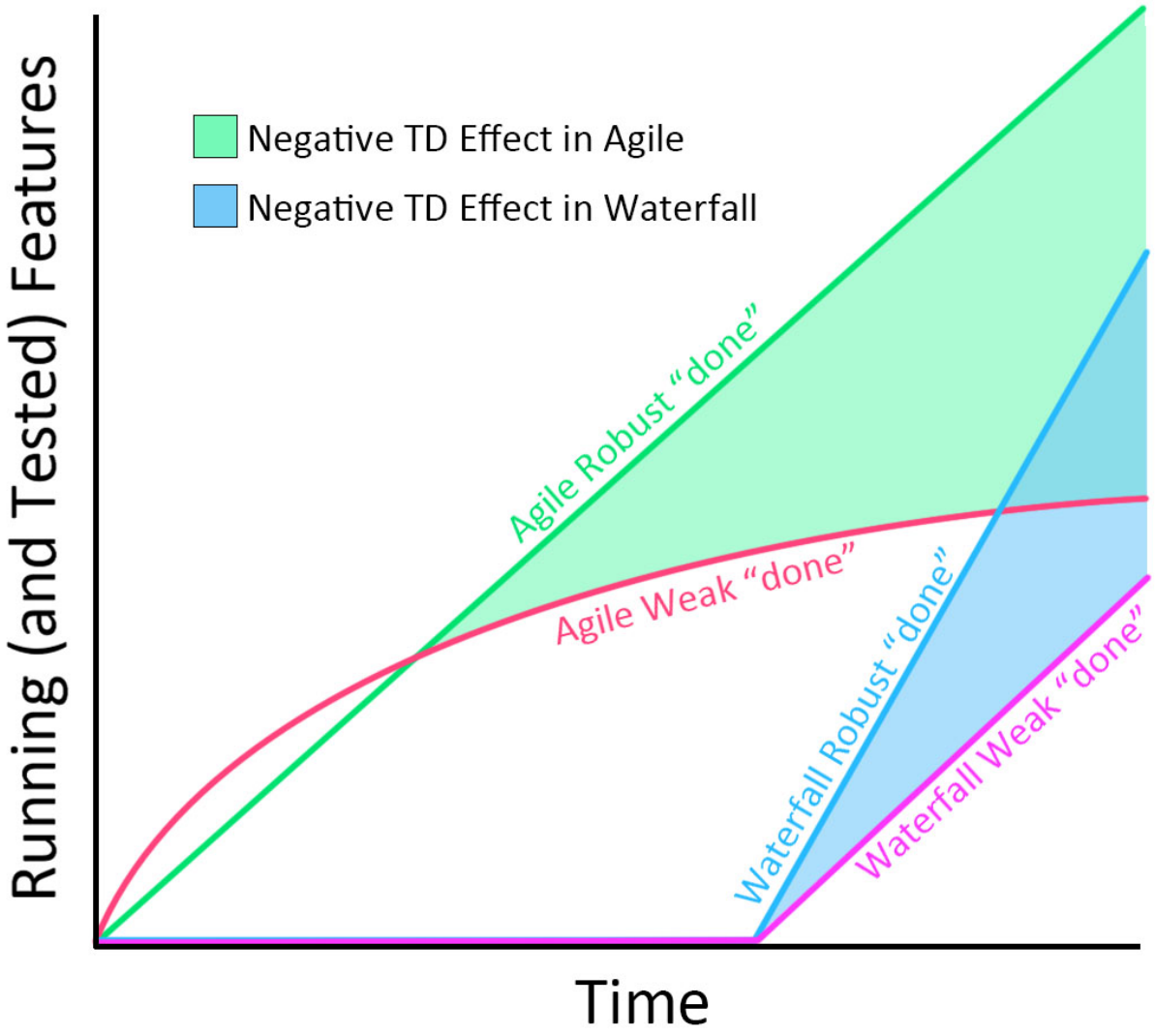
Predicting the Technical Debt (TD) Level in Early Software Development Lifecycle

Qiao Zhang
Advisor: LiGuo Huang

Dept. of Computer Science and Engineering
Southern Methodist University
{qiaoz, lghuang}@smu.edu

SERC Doctoral Student Forum

Technical Debt (TD) Management in Agile



Early Life Cycle TD Management: Challenge or Opportunity?

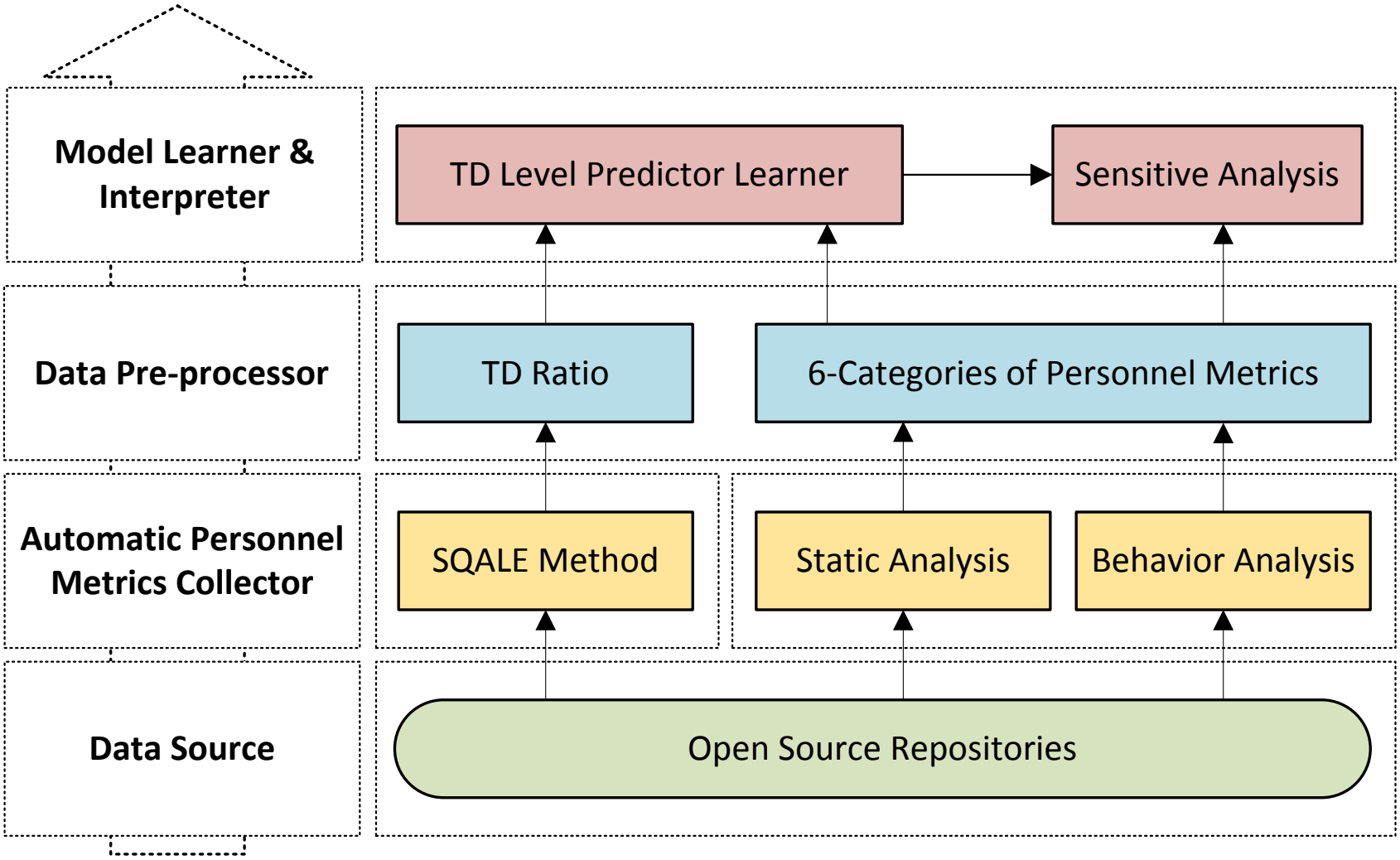
- **Intentional TD** can create big return on investment (ROI) if well managed.
- The sooner we can detect **unintentional TD**, the better we can manage them.
- **Resource is limited.** We should prioritize TD management tasks and strategically allocate resources.
- Project managers tend to overlook the TD management at early software system development lifecycle because
  - —TD issues are often manifested later in the development lifecycle due to the accumulation of TD interest.
  - —TD is difficult to measure at the early lifecycle of system development due to the lack of information.

TD Level Prediction in Early Lifecycle

- The available information is very limited at early lifecycle:
  - Personnel (developer) allocation of the current project
  - Personnel (developer) performance of previous projects
  - Personnel (developer) general evaluation (or peer review)

- This research proposes a learning-based approach to develop an **Early Lifecycle Technical Debt Predictor (ELTDP)** to predicts the level of TD for a software system project based on **six-categories** of personnel (developer)-related metrics that can be collected at the early system development lifecycle.
  - Programming Capability
  - Design Capability
  - Communication Capability
  - Recognition Level
  - Activeness Level
  - Impact Level

ELTDP Learner Overview



SQALE Method

- The **SQALE method** (Software Quality Assessment based on Life Cycle Expectations) is developed for assessing the quality of source code, particularly for the evaluation of the TD during software development

$$Debt = duplication + voilations + comments + coverage + complexity + design$$

Where

$$duplication = cost\_to\_fix\_one\_block \times duplicated\_blocks$$

$$violations = cost\_to\_fix\_one\_violation \times mandatory\_violations$$

$$comments = cost\_to\_comment\_one\_API \times \text{public\_undocumented\_API}$$

$$coverage = cost\_to\_cover\_one\_of\_complexity \times uncovered\_complexity\_by\_tests$$

$$design = cost\_to\_cut\_an\_edge\_between\_two\_files \times package\_edges\_weight$$

1. Letouzey, J. L. (2012, June). The SQALE method for evaluating technical debt. In Proceedings of the Third International Workshop on Managing Technical Debt (pp. 31-36). IEEE Press.

2. Griffith, I., Reimanis, D., Izurieta, C., Codabux, Z., Deo, A., & Williams, B. (2014, September). The correspondence between software quality models and technical debt estimation approaches. In *Managing Technical Debt (MTD), 2014 Sixth International Workshop on* (pp. 19-26). IEEE.

3. Tomas, P., Escalona, M. J., & Mejias, M. (2013). Open source tools for measuring the Internal Quality of Java software products. A survey.Computer Standards & Interfaces, 36(1), 244-255.

Learning Objective: Technical Debt Ratio

$$Debt\ Ratio* = Technical\ Debt / Estimated\ Development\ Cost = TD{\downarrow}index \times 8 \times 60 / LOC \times Cost{\downarrow}per\ line$$

Where,

- $TD{\downarrow}inex$ is estimated based on SQALE method in person days,
- 8×60 represents 8 working hours per day and 60 minutes per hour,
- LOC represents the (estimated) number of lines of code of current project,
- $Cost\ per\ line$ represents the (estimated) time to develop one line of code in minutes.

- **Debt Ratio** measures the ratio between the actual TD and the effort it would take to rewrite the entire system code from scratch.

- The **SQALE TD Rating** is determined based on the Debt Ratio (e.g., less than 2% as A. 2% to 8 % as B. and so on).

* Formula and interpretation are from SonarQube Documentation – User Guide – Technical Debt created by Ann Campbell on Jan 11, 2016. (Link: http://docs.sonarqube.org/display/SONARQUBE52/Technical+Debt#TechnicalDebt-Computationoftechnicaldebt)

6-Categories of 44 Personnel Metrics

	Metric Name	Description	Cat.*
Basic Info	public_repos	No. of public repositories (repos)	PC
	assignees	No. of being assignees of public repos	RL
	follower	No. of follower	RL
Owned Repo Info	analyzed_repos	No. of analyzed repos (ARs)	PC
	avg_size	Avg. size (in byte) of ARs	PC
	avg_stargazers	Avg. stars of ARs	RL
	avg_subscribers	Avg. subscribers of ARs	RL
	avg_forks	Avg. forks of ARs	IL
	avg_issues	Avg. issues of ARs	RL
	avg_handled_issues	Avg. closed issues of analyzed repos	AL
	avg_issue_handled_days	Avg. days between issue creation and first qualified handling event (e.g., close, milestone)	AL
Contributed Repo Info	pull_requests	No. of pull requests (PRs)	AL
	accepted_pull_requests	No. of accepted PRs	RL
	contributed_repos	No. of repos among accepted PRs	RL
	avg_commits	Avg. No. of commits among accepted PRs	IL
	avg_additions	Avg. No. of addition among accepted PRs	IL
	avg_deletions	Avg. No. of deletion among accepted PRs	IL
	avg_changed_files	Avg. No. of changed files among accepted PRs	IL
	avg_days_interval	Average days between two accepted PRs	AL

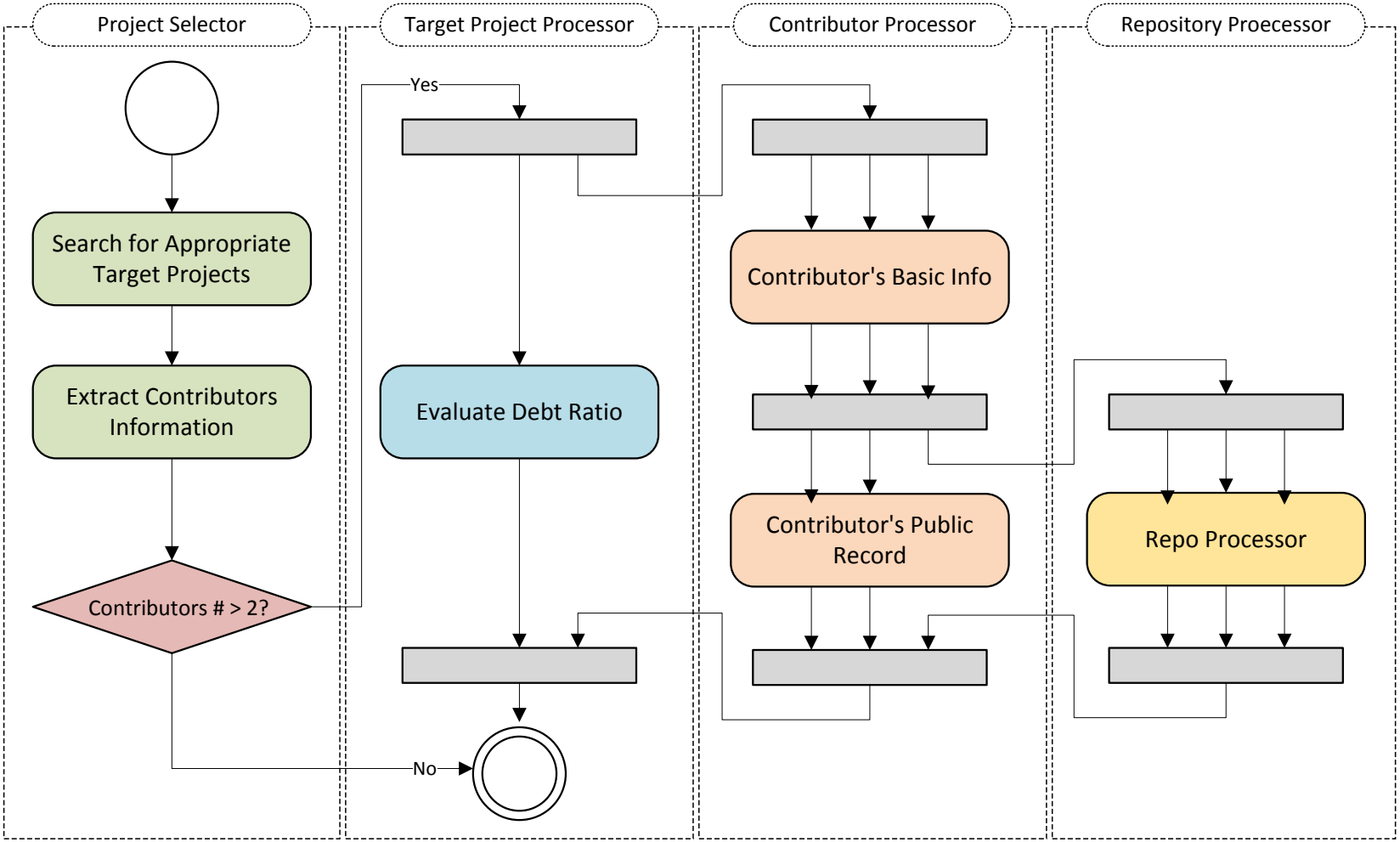
* PC: programming capability; RL: recognition level; AL: activeness level; IL: Impact level

6-Categories of 44 Personnel Metrics (Cont'd)

		Metric Name	Description	Cat.*
Static Analysis of Previous Repositories with SonarQube	technical debt	avg_code_smells	Avg. No. of code smells (TD) of analyzed repos	DC
		avg_sqale_index	Avg. SQALE index (in person days) of analyzed repos	DC
		avg_debt_ratio	Avg. SQALE debt ratio of analyzed repos	DC
	reliability	avg_bugs	Avg. bugs of analyzed repos	PC
	vulnerability	avg_vulnerabilities	Avg. vulnerabilities of analyzed repos	DC
	duplications**	avg_duplicated_lines_density	Avg. duplicated lines ratio of analyzed repos	DC
		avg_duplicated_blocks	Avg. duplicated blocks of analyzed repos	DC
	size**	avg_nloc	Avg. lines of code of analyzed repos	PC
		avg_statements	Avg. statements of analyzed repos	PC
		avg_functions	Avg. functions of analyzed repos	PC
		avg_classes	Avg. classes of analyzed repos	PC
	complexity**	avg_complexity	Avg. cyclomatic complexity of analyzed repos	DC
		avg_function complexity	Avg. complexity by function of analyzed repos	DC
		avg_class complexity	Avg. complexity by class of analyzed repos	DC
	documentation**	avg_comment_line_density	Avg. comment lines of analyzed repos	CC
		avg_pulic_api	Avg. public APIs of analyzed repos	CC
		avg_documented_api_density	Avg. documented API ratio of analyzed repos	CC
		avg_undocumented_api	Avg. undocumented APIs of analyzed repos	CC

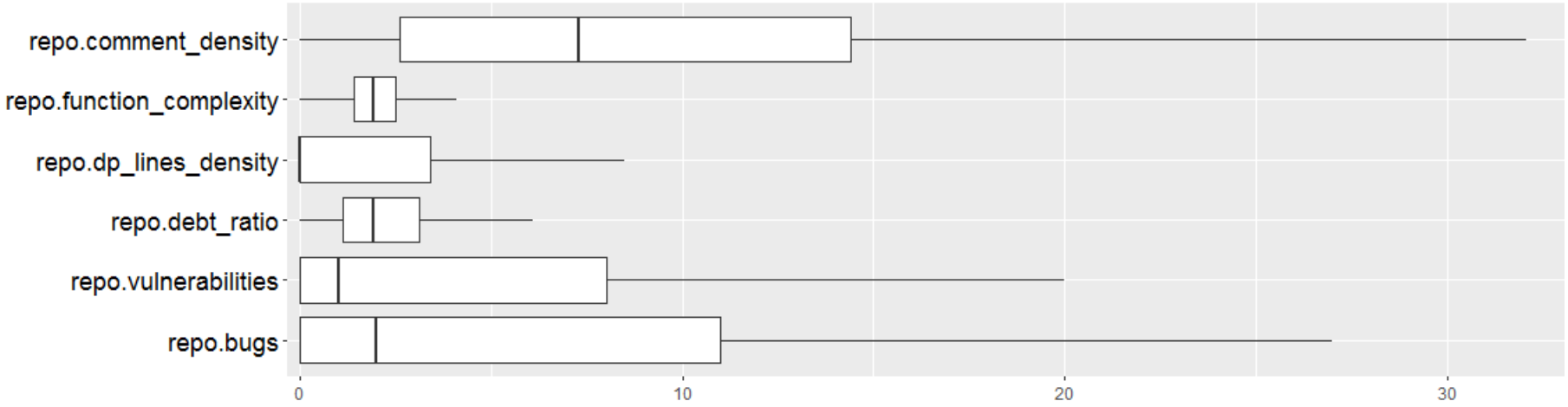
* PC: programming capability; DC: design capability; CC: communication capability; ** Not a completed set

Automatic Personnel Metrics Collection



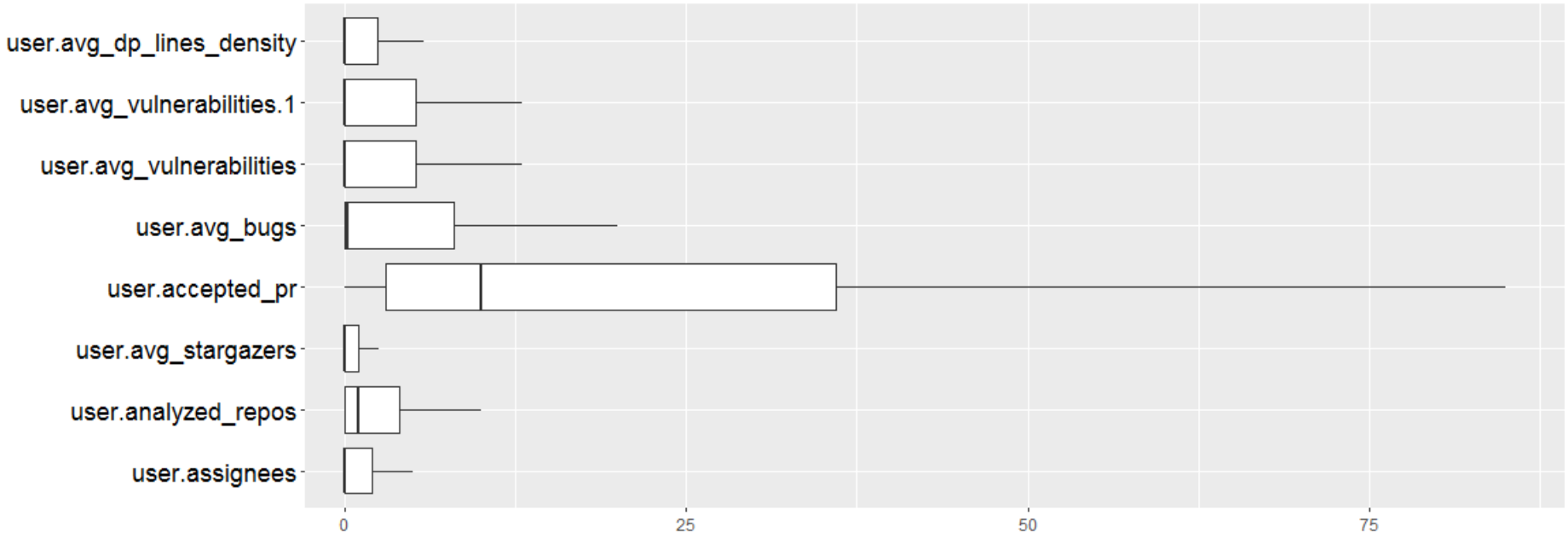
Data Overview: Open-Source Repositories

- We have explored
  - 933 open source projects
  - 11,258 developers (contributors of target projects)
  - 35,112 repositories (historical projects in which developers were involved)



Overview of partial attributes of investigated repositories

Data Overview: Contributors



Overview of partial attributes of investigated contributors

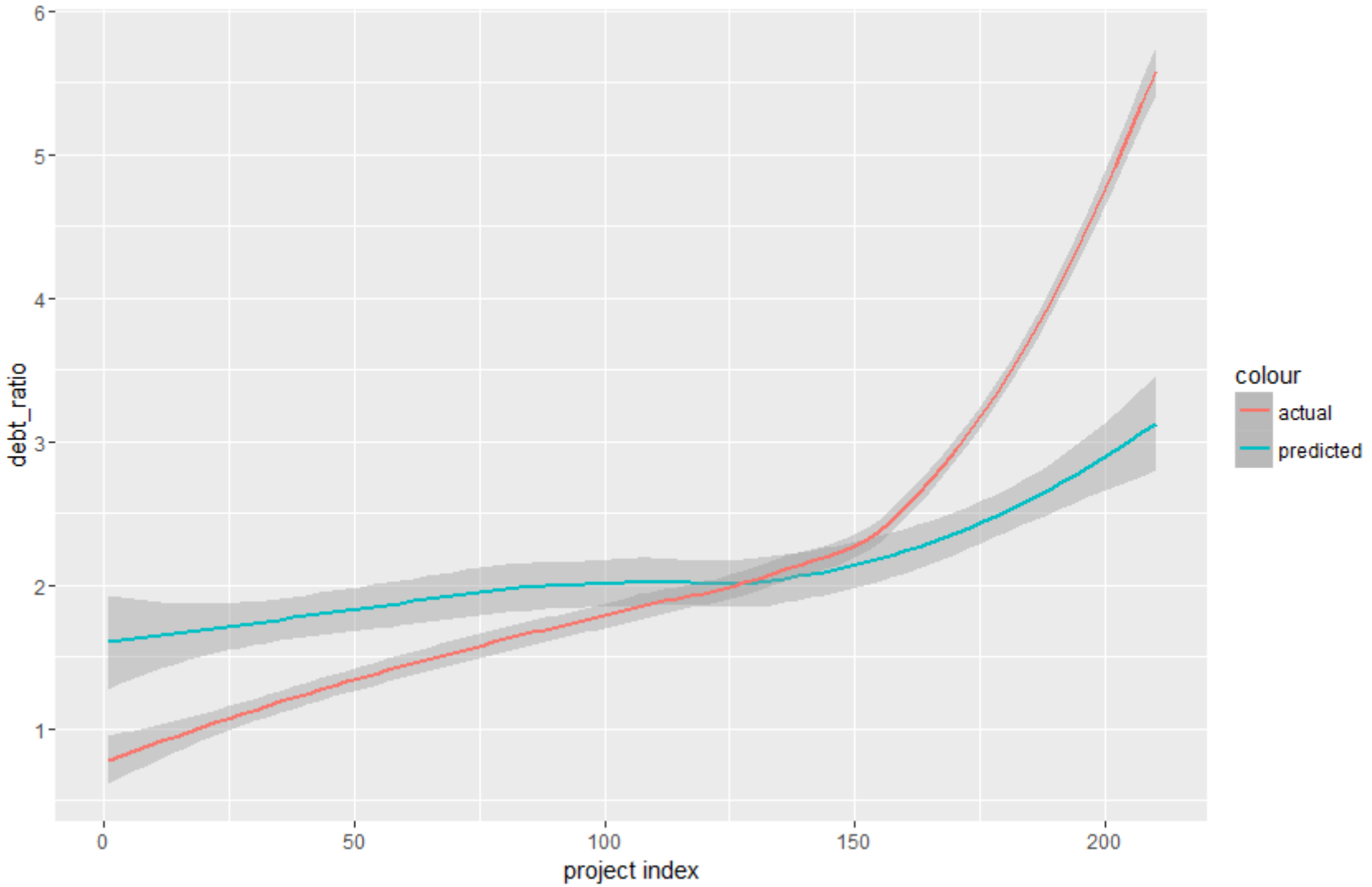
- The automated personnel metrics collection tool will be integrated into the ELTDP Learner.
- The selection of target projects is currently based on language, size, and number of forks of open source project repositories.

Initial Experiment Results

Algorithm	Radial Basis Function Regression with SVM (Support Vector Machine)
Actual Debt Ratio Range	[0.6 , 8.5]
Correlation coefficient	0.5198
Mean absolute error	0.7104
Root mean squared error	1.0097
Relative absolute error	87.4951 %
Root relative squared error	85.6002 %

- We use 10 fold cross-validation with randomly selected 80% percent of collected data as training set and rest 20% as testing set.
- The ELTDP Learner will find the most accurate prediction model among a number of machine learning algorithms.
- There will be a sensitive analysis of the learned prediction model for better interpretation.
- If the accuracy of prediction were acceptable, the predicted TD Ratio could be applied directly on the cost estimation results.

Actual and Predicted Debt Ratio



Conclusions and Future Work

- This research aims at predicting TD Level at the early lifecycle of software system development.
- We propose a learning-based approach to develop an Early Lifecycle TD Predictor (ELTDP) based on six categories of personnel metrics that can be collected once the development team is formed.
- A prototype of an automatic personnel metrics collection tool has been developed.
- **Ultimate goal:** A tool package that integrates data collection, prediction model learning, as well as learned model interpretation functions to facilitate project managers to make early decision on the strategies of TD management.

Thank you!

Qiao Zhang
Advisor: LiGuo Huang

Dept. of Computer Science and Engineering
Southern Methodist University
{qiaoz, lghuang}@smu.edu

SERC Doctoral Student Forum