A Systematic Study on Empirical Software Engineering Research in 2018: Categorizing

陈俊达，张傲，刘瑗玮

Software Institute, NJU

April 23, 2019

Table Of Contents

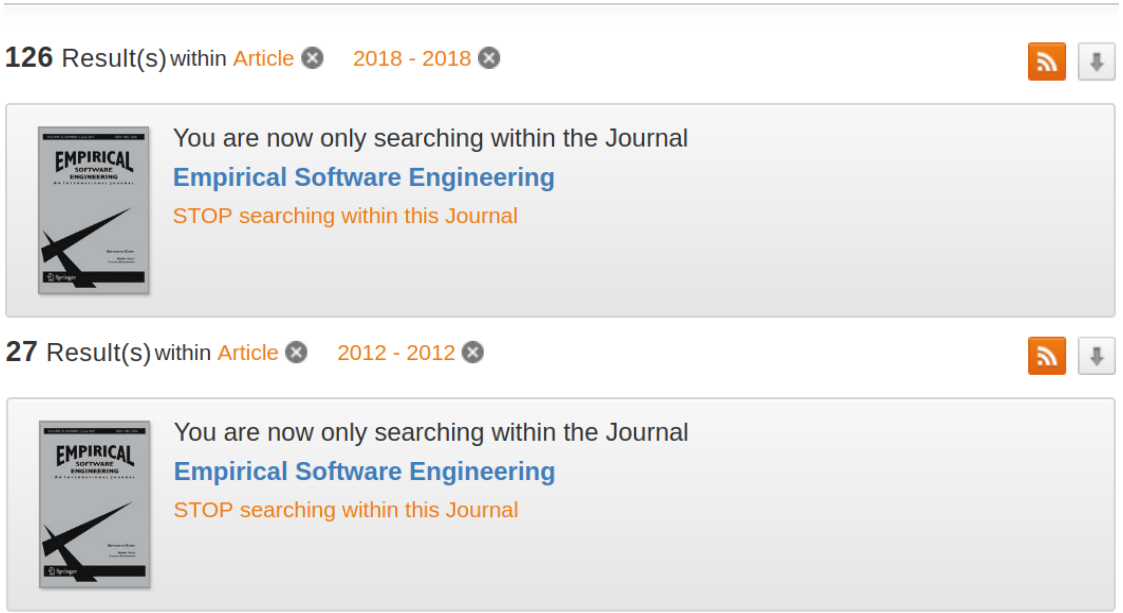
Contributions

In this presentation:

- We show the **difficulty** to analyze vast articles.
- We develop TAC-IE, an **efficient** approach to classify papers amongst vast articles.
- We propose our **criteria** to identify different research approaches.
- We present our **statistical result** on 180 papers from three data sources published in 2018.

Difficulty

年份选得好，作业似高考……



Papers:

- 2018: 180 total
- 2012: 73 total

TAC-IE:Efficient paper reading technique

一篇好的论文一般都会遵循如下结构：

- **T**itle
- **A**bstract
- **I**ntroduction
- Related work & background
- Research & **E**xperiment
- Result
- **C**onclusion

我们的方法：

- 首先
  - 观察标题 (**T**)、摘要 (**A**)、结论 (**C**)，大致了解论文要解决的问题与实验方式
  - 初步确定
- 其次
  - 观察介绍 (**I**) 的最后两段，和实验设计部分 (**E**)
  - 确定实验方法

举个栗子[1]

Decision making and visualizations based on test results

ABSTRACT

Background: Testing is one of the main methods for quality assurance in the development of embedded software, as well as in software engineering in general. Consequently, test results (and how they are reported and visualized) may substantially influence business decisions in software-intensive organizations. **Aims:** This case study examines the role of test results from automated nightly software testing and the visualizations for decision making they enable at an embedded systems company in Sweden. In particular, we want to identify the use of the visualizations for supporting decisions from three aspects: in daily work, at feature branch merge, and at release time. **Method:** We conducted an embedded case study with multiple units of analysis by conducting interviews, questionnaires, using archival data and participant observations.

[1] P.E. Strandberg et al. (2018). Decision making and visualizations based on test results. In Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '18).

3.2 Data Collection

In order to approach the research questions from different angles and to ensure completeness of data collection, we used methodological triangulation by using multiple data collection techniques: interviews, questionnaires (as follow-up to interviews), archival data and participant observations. An overview is shown in Figure 4 and details explained below.

Interviews: We conducted a total of 5 semi-structured interviews with one project manager, one test manager, two developers and one test environment developer. We selected these persons based on convenience sampling as the industrial co-author of this paper, in consultation with the test manager, knew the most knowledgeable persons to interview in the company. We planned a series of questions before the interview but did not necessarily ask them in the exact order every time, but made sure that every question was asked. We started each interview with background questions about the interviewee, before asking specific questions related to our research objective. Two authors participated in each of the 5 interviews and also took personal notes. The interviewees had access to the test results visualization web page while they answered questions, and we encouraged them to use it during the interview, such that references to particular pages and figures can be easily made. The interview questions are given in Appendix A.

Questionnaires: We followed the interviews with a web-based questionnaire to the 5 interviewees. The questionnaire was kept small and included questions on their degree of satisfaction with existing visualization support for making decisions in their daily work, at branch merge and at release time. Specific questions were asked about ease of understandability of visualizations, ease of learning the meaning, time taken to visualize and ease of navigation. Each question had an optional space for including any free-text comments. The used questionnaire is given in Appendix B.

The questionnaires were given to the same persons we interviewed to serve two purposes: (i) to complete and complement the information given in interviews (ii) to be able to elicit usefulness of test visualizations, their ease of navigation, learnability, understandability and response times.

Archival Data: We had access to internal documentation as well as source code repositories from past and present projects. Documents such as formal process definitions and more informal internal wiki pages were also available. The archival information was mainly used to get familiar with Westermo's development process, the meaning of various internal classifications of test results as well as getting an overview of the database schema, associated script documentation and generated visualizations. This was essential to confirm, understand and add to the data collected from interviews and questionnaires and also helped us answer *RQ1*.

Participant Observations: Twice a week at Westermo a team of people with different roles gather around a large screen in a conference room for a sync meeting where test results are investigated by using the TRDB and the visualizations. The results are discussed and tasks of various kind, typically fault finding, are distributed. This is an important ritual for sharing knowledge like "Test system X is new and the devices in the system are early prototypes." We participated in two sync meetings and made high-level notes on how the visualizations were used and the shortcomings encountered.

Participant Observations. Twice a week at Westermo a team of people with different roles gather around a large screen in a conference room for a sync meeting where test results are investigated by using the TRDB and the visualizations. The results are discussed and tasks of various kind, typically fault finding, are distributed. This is an important ritual for sharing knowledge like "Test system X is new and the devices in the system are early prototypes." We participated in two sync meetings and made high-level notes on how the visualizations were used and the shortcomings encountered.

Five Research Methods

五大类研究方法：ECSEA

- Experiment
- Case Study
- Survey
- Ethnography
- Action Research

我们的分类方法：

- **Characteristics+Keywords**

Example: Experiment

- 出现 experiment 关键字
- 强调了控制变量
- 提出新方法: 新方法常常要和之前方法对比，以证明优越性
- quasi-experiment

A controlled experiment on time pressure and confirmation bias in functional software testing

Iflaah Salman[1] · Burak Turhan[2] · Sira Vegas[3]

Published online: 18 December 2018

Abstract

Context Confirmation bias is a person's tendency to look for evidence that strengthens his/her prior beliefs rather than refutes them. Manifestation of confirmation bias in software testing may have adverse effects on software quality. Psychology research suggests that time pressure could trigger confirmation bias.

Objective In the software industry, this phenomenon may deteriorate software quality. In this study, we investigate whether testers manifest confirmation bias and how it is affected by time pressure in functional software testing.

Method We performed a controlled experiment with 42 graduate students to assess manifestation of confirmation bias in terms of the conformity of their designed test cases to the provided requirements specification. We employed a one factor with two treatments between-subjects experimental design.

Results We observed, overall, participants designed significantly more confirmatory test cases as compared to disconfirmatory ones, which is in line with previous research. However, we did not observe time pressure as an antecedent to an increased rate of confirmatory testing behaviour.

Conclusion People tend to design confirmatory test cases regardless of time pressure. For practice, we find it necessary that testers develop self-awareness of confirmation bias and counter its potential adverse effects with a disconfirmatory attitude. We recommend further replications to investigate the effect of time pressure as a potential contributor to the manifestation of confirmation bias.

Keywords Test quality · Software testing · Cognitive biases · Confirmation bias · Time pressure · Experiment · Human factors

Example: Case Study

- 出现 case study 关键字
- 一定要有具体 case(某个具体公司 or 项目……)

High-level software requirements and iteration changes: a predictive model

Kelly Blincoe[1] · Ali Dehghan[2] · Abdoul-Djawadou Salaou[2] · Adam Neal[3] · Johan Linaker[4] · Daniela Damian[2]

Published online: 15 October 2018

Abstract
Knowing whether a software feature will be completed in its planned iteration can help with release planning decisions. However, existing research has focused on predictions of only low-level software tasks, like bug fixes. In this paper, we describe a mixed-method empirical study on three large IBM projects. We investigated the types of iteration changes that occur. We show that up to 54% of high-level requirements do not make their planned iteration. Requirements are most often pushed out to the next iteration, but high-level requirements are also commonly moved to the next minor or major release or returned to the product or release backlog. We developed and evaluated a model that uses machine learning to predict if a high-level requirement will be completed within its planned iteration. The model includes 29 features that were engineered based on prior work, interviews with IBM developers, and domain knowledge. Predictions were made at four different stages of the requirement lifetime. Our model is able to achieve up to 100% precision. We ranked the importance of our model features and found that some features are highly dependent on project and prediction stage. However, some features (e.g., the time remaining in the iteration and creator of the requirement) emerge as important across all projects and stages. We conclude with a discussion on future research directions.

Keywords Software requirements · Completion prediction · Release planning · Mining software repositories · Machine learning

Example: Survey

- 出现 survey 关键字
- interview
- questionnaire
- Literature survey: systematic (literature) review, meta-analysis

High-level software requirements and iteration changes: a predictive model

Kelly Blincoe[1] · Ali Dehghan[2] · Abdoul-Djawadou Salaou[2] · Adam Neal[3] · Johan Linaker[4] · Daniela Damian[2]

Published online: 15 October 2018

Abstract
Knowing whether a software feature will be completed in its planned iteration can help with release planning decisions. However, existing research has focused on predictions of only low-level software tasks, like bug fixes. In this paper, we describe a mixed-method empirical study on three large IBM projects. We investigated the types of iteration changes that occur. We show that up to 54% of high-level requirements do not make their planned iteration. Requirements are most often pushed out to the next iteration, but high-level requirements are also commonly moved to the next minor or major release or returned to the product or release backlog. We developed and evaluated a model that uses machine learning to predict if a high-level requirement will be completed within its planned iteration. The model includes 29 features that were engineered based on prior work, interviews with IBM developers, and domain knowledge. Predictions were made at four different stages of the requirement lifetime. Our model is able to achieve up to 100% precision. We ranked the importance of our model features and found that some features are highly dependent on project and prediction stage. However, some features (e.g., the time remaining in the iteration and creator of the requirement) emerge as important across all projects and stages. We conclude with a discussion on future research directions.

Keywords Software requirements · Completion prediction · Release planning · Mining software repositories · Machine learning

Example: Ethnography

- 参与式观察 local people 的直观感受

ABSTRACT

Background: Testing is one of the main methods for quality assurance in the development of embedded software, as well as in software engineering in general. Consequently, test results (and how they are reported and visualized) may substantially influence business decisions in software-intensive organizations. **Aims:** This case study examines the role of test results from automated nightly software testing and the visualizations for decision making they enable at an embedded systems company in Sweden. In particular, we want to identify the use of the visualizations for supporting decisions from three aspects: in daily work, at feature branch merge, and at release time. **Method:** We conducted an embedded case study with multiple units of analysis by conducting interviews, questionnaires, using archival data and participant observations. **Results:** Several visualizations and reports built on top of the test results database are utilized in supporting daily work, merging a feature branch to the master and at release time. Some important visualizations are: lists of failing test cases, easy access to log files, and heatmap trend plots. The industrial practitioners perceived the visualizations and reporting as valuable, however they also mentioned several areas of improvement such as better ways of visualizing test coverage in a functional area as well as better navigation between different views. **Conclusions:** We conclude that visualizations of test results are a vital decision making tool for a variety of roles and tasks in embedded software development, however the visualizations need to be continuously improved to keep their value for its stakeholders.

Example: Action Research

- 多次迭代
- 每次迭代都有反馈调整
- 问题提出者参与比较少
- 只通过 abstract 容易误判，需要读内容才能判断

ABSTRACT

Background: The vision of a fourth industrial revolution lately strongly captured the attention of research A Cyber-physical system (CPS) is one of the main drivers of this vision. Such system controls an underlying factory interacting with sensors, actuators and other systems creating systems-of-systems. A main point of interest is how these components are built and interconnected, i.e. the system's architecture, and how it might be improved to increase reliability and security.

Aims: Unfortunately, there is no review available describing and identifying recent research status and progress for such architectures. Thus, in this systematic mapping study, we overview research on architecture for CPS in the context of Industry 4.0.

Method: With an initial automatic search and through iterative refining, first results were gathered. Next, forward and backwards snowballing was performed to integrate the results, giving a final population of 213 papers. The output of this study is firstly a categorization and plot of architectural styles. Secondly, the categorization is extended to include security concerns.

Results: In general, there is a tendency in proposing solutions in the fields of system and software architecture while other areas are visibly under-researched. Most proposals focus in digital representation, information management and integration of solutions.

Conclusions: While a lot of solution proposals for different architectures exist, there are many fields uncovered and most of the solutions still lack of validation and evaluation results. Our findings highlight areas for future research and provide suggestions for

Example: 一种误判范例

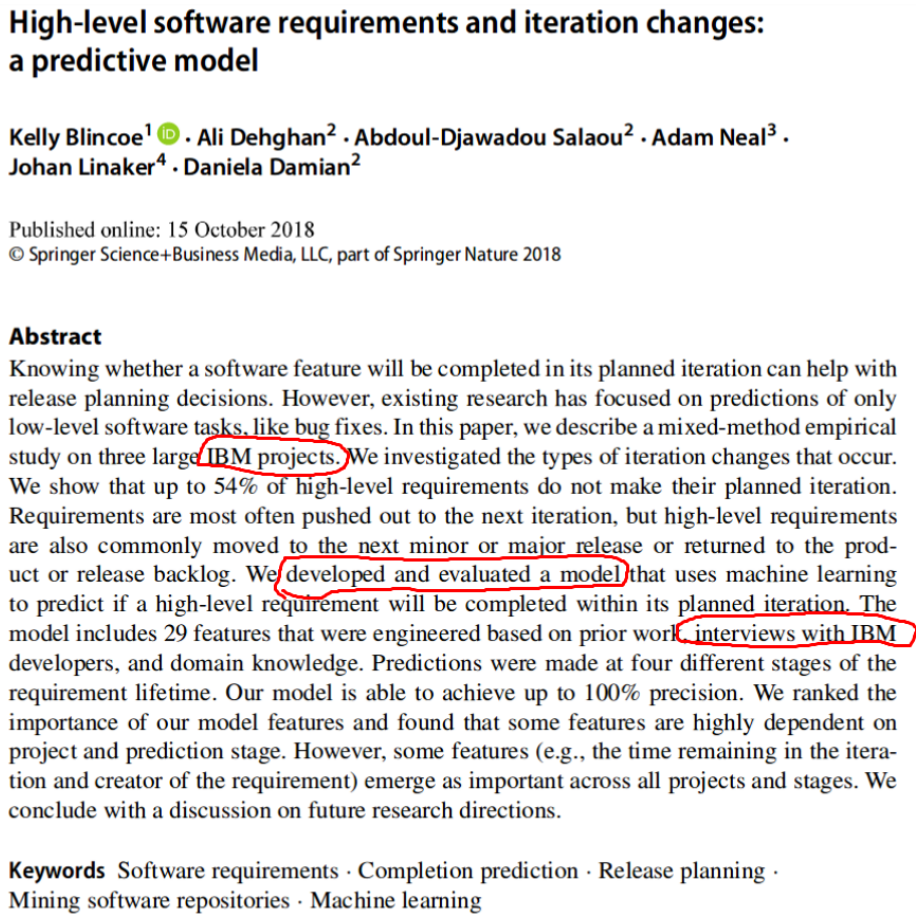
5.2 Iterations

The execution of the case study consists in a set of iterations, which follow the template structure outlined in Section 4. Each iteration is aimed at answering one or more RQs, and, although the overall case study is exploratory, each iteration has a different flavour, which range from *exploratory*, to *explanatory* and to *improving*. Furthermore, in each iteration, different tasks of the template are performed. Tables 5 and 6 give an outline of the different iterations. Overall, the case study consists of six iterations. The first one is a *Pilot Study*, based on a preliminary requirements dataset (*D-Pilot*), while the others belong to the *Large-scale Study*, based on a larger requirements dataset (*D-Large*). Table 5 shows the nature of the iteration, the associated RQs, the patterns and dataset used. Iterations from 0 to 2 were dedicated to investigate the accuracy of NLP patterns (RQ1, RQ2), with different levels of insight. Iteration 3 was dedicated to improve the precision of the patterns (RQ3). Iteration 4 and 5 were focused on the application of the SREE dictionaries (RQ4.1-4). Table 6 shows the tasks performed together with the subjects who participated to the task. The notation VE1/VE2 indicates that the task, initially conducted by VE1, was replicated by VE2.

Here, we briefly summarise the rationale, execution and results of each iteration, with reference – explicit or implicit – to Table 5 and 6. We do not provide all the justifications for the content of the tables, since extensive details are given in the subsequent sections.

Example: Hybrid Method

- experiment
- case study
- survey



High-level software requirements and iteration changes: a predictive model

Kelly Blincoe[1] · Ali Dehghan[2] · Abdoul-Djawadou Salaou[2] · Adam Neal[3] · Johan Linaker[4] · Daniela Damian[2]

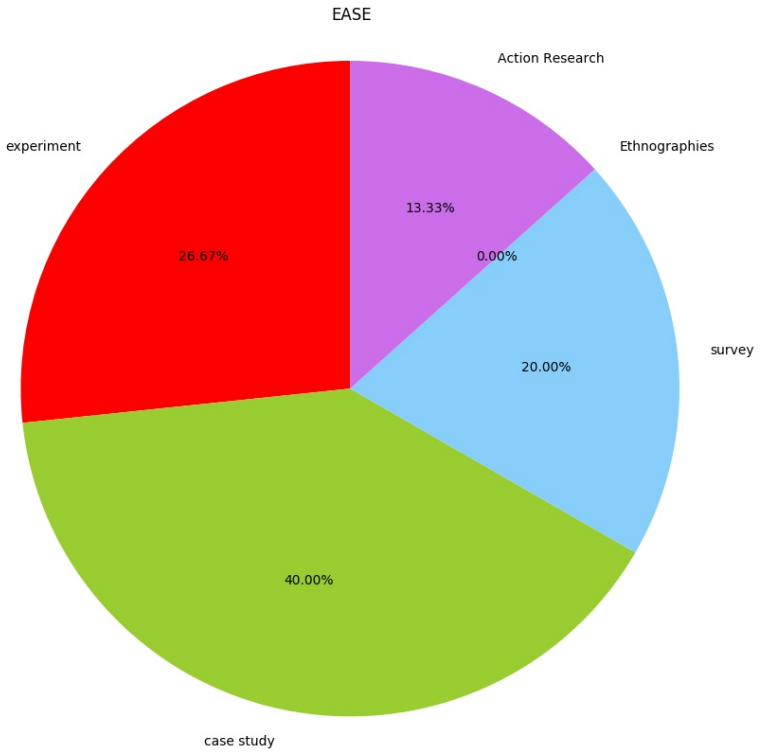
Published online: 15 October 2018

Abstract
Knowing whether a software feature will be completed in its planned iteration can help with release planning decisions. However, existing research has focused on predictions of only low-level software tasks, like bug fixes. In this paper, we describe a mixed-method empirical study on three large IBM projects. We investigated the types of iteration changes that occur. We show that up to 54% of high-level requirements do not make their planned iteration. Requirements are most often pushed out to the next iteration, but high-level requirements are also commonly moved to the next minor or major release or returned to the product or release backlog. We developed and evaluated a model that uses machine learning to predict if a high-level requirement will be completed within its planned iteration. The model includes 29 features that were engineered based on prior work, interviews with IBM developers, and domain knowledge. Predictions were made at four different stages of the requirement lifetime. Our model is able to achieve up to 100% precision. We ranked the importance of our model features and found that some features are highly dependent on project and prediction stage. However, some features (e.g., the time remaining in the iteration and creator of the requirement) emerge as important across all projects and stages. We conclude with a discussion on future research directions.

Keywords Software requirements · Completion prediction · Release planning · Mining software repositories · Machine learning

Frequencies[2]-EASE

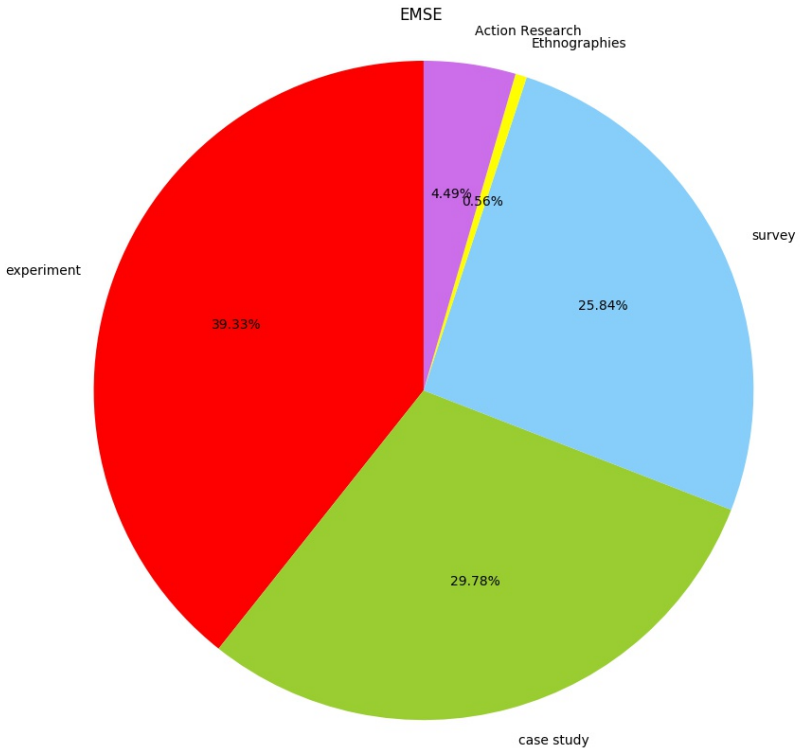
- experiment
- case study
- survey
- ethnography
- action research



[2]* with respect to total method appearances

Frequencies[3]-EMSE

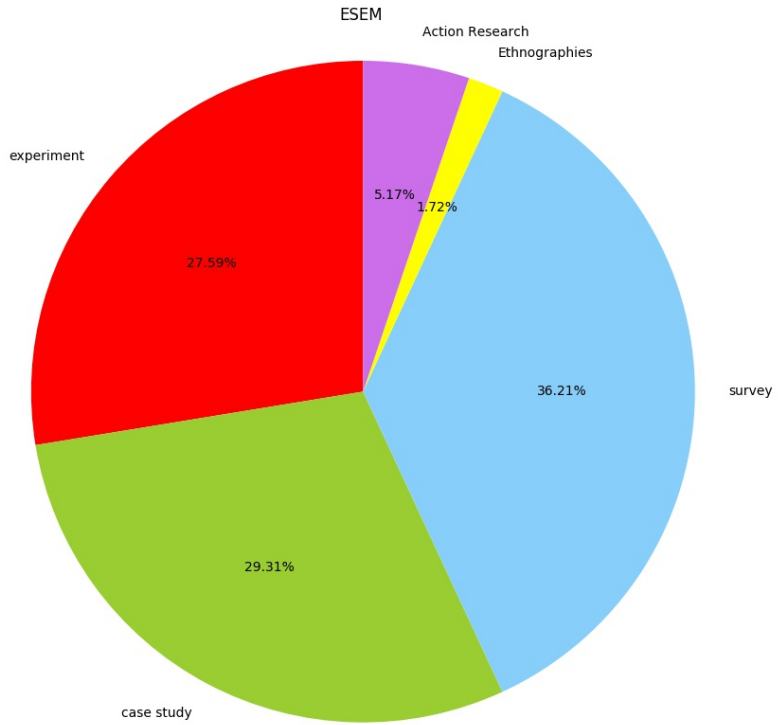
- experiment
- case study
- survey
- ethnography
- action research



[3]* with respect to total method appearances

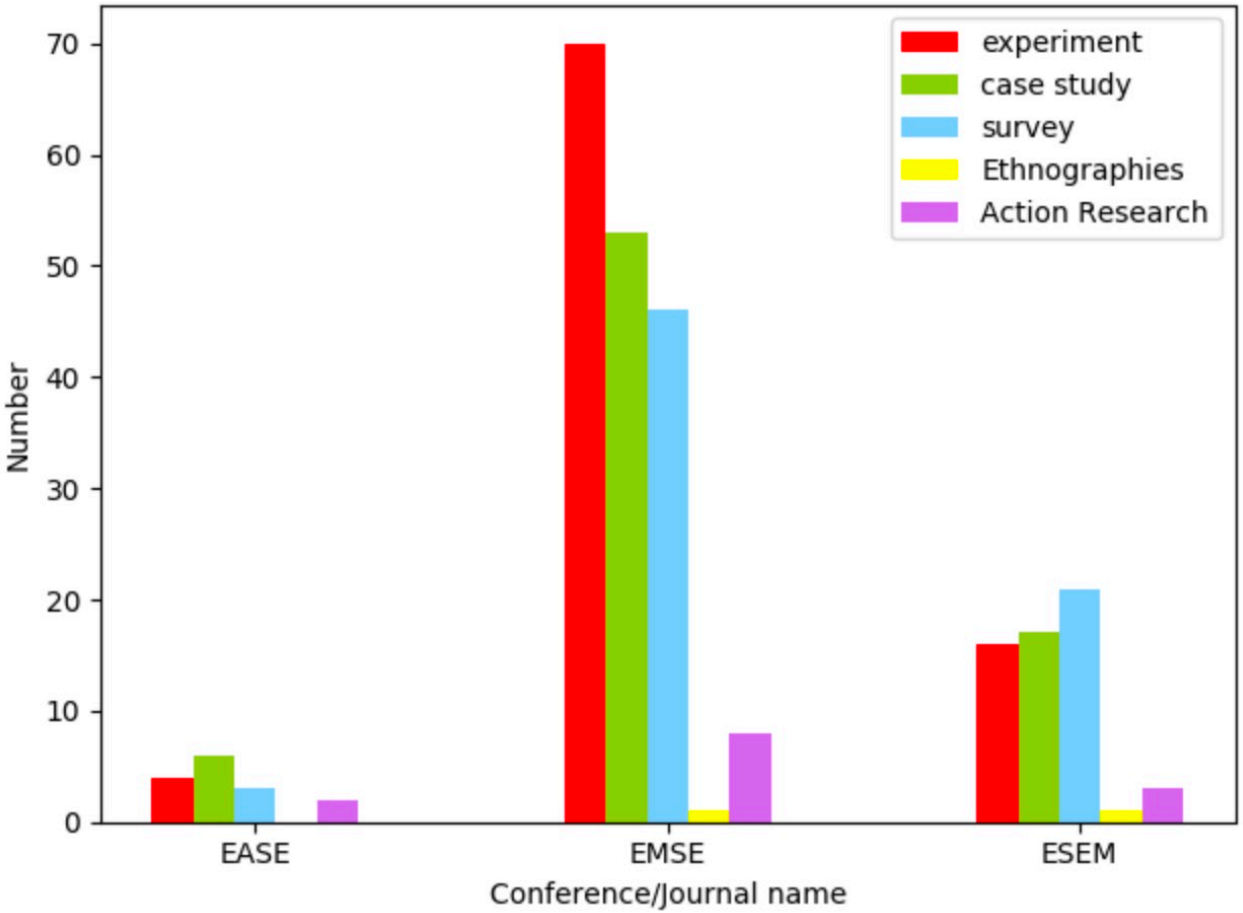
Frequencies[4]-ESEM

- experiment
- case study
- survey
- ethnography
- action research



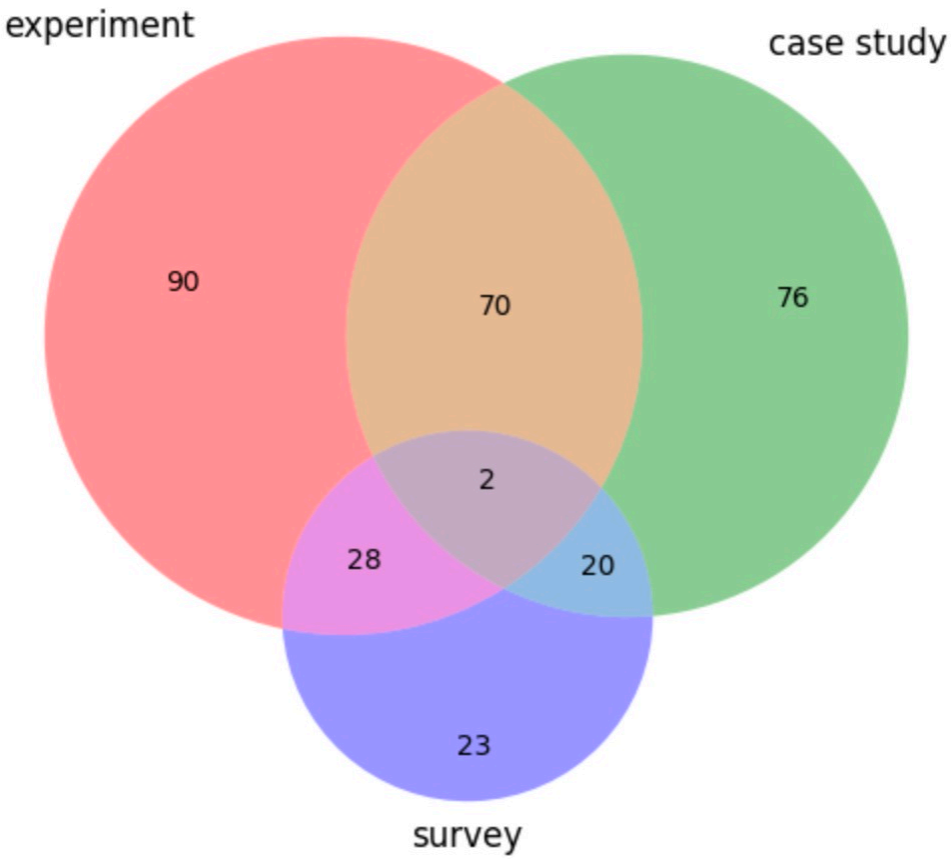
[4]* with respect to total method appearances

Overview-1[5]



[5] Papers with hybrid methods included

Overview-2[6]



[6]Consider only the top-3 frequent methods

- **Experiment**,**Case study**,**Survey** 是最常用的三种方法
- 采用多种方法的研究并不罕见

Acknowledgements

- 任务平均分配，所有组员都参与了文献的阅读之中。
  - 王瑞华，张　傲，连远翔:EMSE1-60
  - 杨宇清，雷　诚，周康桥:EASE1-12,EMSE61-108
  - 刘瑗玮，陈俊达，张翔宇:ESEM1-42,EMSE109-126
- 本报告提出的所有方法与实验结论均为小组讨论成果。
- 演讲者：陈俊达，张傲，刘瑗玮
- 幻灯片与报告: 杨宇清
- 图表：刘瑗玮

A Systematic Study on Empirical Software Engineering Research in 2018: Experiment

雷诚，连远翔，杨宇清

Software Institute, NJU

May 10, 2019

Table Of Contents

Contributions

In this presentation:

- We present three different examples of interesting experiments
- We show that experiments may take various forms, and we discuss the reasons.
- We identify and distinguish the difference between the three experiments against the standard ones.
- We show the process of double-checking and correcting previous results.

Arrangements

We organize the examples into two groups:

- performances of algorithm on computer.
  - ”our method is better”
  - Solidity of algorithm
- performance of approaches upon people.
  - Uncertain
  - Erasing bias
  - ”quasi-experiment”

Therefore the examples are arranged as:

- Within-experiments of algorithm
- Between-experiments of algorithm
- A quasi-experiment of people

Idea

Provide an approach to reduce the compilation time of test code

Experiment Unit

- Test cases from EEMBC and SPEC programs

Subject	Domain	Description	#Tests
a2time01	EEMBC - Automotive	Angle-to-time conversion	10K
aifftr01	EEMBC - Automotive	Fast Fourier transforms	10K
aifirf01	EEMBC - Automotive	Finite Impulse Response filter	10K
aiifft01	EEMBC - Automotive	Inverse Fast Fourier transforms	10K
rspeed01	EEMBC - Automotive	Road speed calculation	10K
autcor00	EEMBC - Telecom	Cross correlation of signals	10K
conven00	EEMBC - Telecom	Convolutional encoding	10K
fbital00	EEMBC - Telecom	Bit allocation	10K
fft00	EEMBC - Telecom	Fast Fourier transforms	10K
viterb00	EEMBC - Telecom	Viterbi decoding	10K
401.bzip2	SPEC - Integer	Compression	10K
462.libquantum	SPEC - Integer	Quantum computing	10K
444.namd	SPEC - Floating Point	Molecular dynamics simulation	10K
470.lbm	SPEC - Floating Point	Computational fluid dynamics	10K
999.specrand	SPEC - Floating Point	Pseudo-random number generation	10K
bufferTS	ComputeCPP	Arithmetic operations on the cl::sycl::buffer class	10K
imageTS	ComputeCPP	Arithmetic operations on the cl::sycl::image class	10K

Table 1: Subject programs used in our experiment

Experiment Unit: Source

- 15 applications
- from 2 industrial test suite
- plus an industry program

In this Section, we describe the programs and associated tests used in our experiment. We used 15 subject programs from 2 industry standard benchmark suites, EEMBC and SPEC, that cover a wide range of applications. We also evaluate our approach using an industry provided program, ComputeCPP, developed at Codeplay. Subject programs in EEMBC and SPEC benchmarks were accompa-

Treatment

Methods. We propose transformations that reduce the number of instructions in the test code, which in turn reduces compilation time. Using two well known compilers, GCC and Clang, we conduct empirical evaluations using subject programs from industry stan-

- Apply the convert algo or not
- GCC+Clang

Input: *TC* test code, *PF* program function
Output: *TTC* transformed test code
1: Create a copy of *TC*, call it *TTC*.
2: Search *TTC* for parameterized test suites, and record all calls of *PF* and its input arguments.
3: Merge the input data of the same type from all the tests into centralized data structures *DS*.
4: Merge the multiple *PF* calls into a single *PF* call embedded in a loop with as many iterations as there are tests in the parameterized test suite.
5: Update the *PF* call within the loop so that it accepts the correct slice of data from *DS*.
6: Return *TTC*.

Algorithm 1: Test code transformation

Response

dard benchmarks and an industry provided program. We evaluate *compilation speedup*, *execution time*, *scalability* and *correctness* of the proposed test code transformation.

Q1. Compilation Speedup: *Does the proposed transformation, relative to existing compiler optimisations, speedup test code compilation?* To answer this question, we used test suites of varying sizes, from 10 to 10K tests, for each subject program and measured the compilation times before and after the transformation, enabling all existing compiler optimisations.

Q2. Execution: *Does the transformation slow down execution of the test code?* To examine this question, for each program and associated test suite, we compare running times of the original and transformed versions of the test code.

Q3. Correctness: *Does the transformation preserve correctness of test executions?* For each benchmark and associated test suite, we compared values of internal states and outputs, obtained during execution of each of the tests in the suite with the original and transformed test code.

Q4. Scalability *Does the transformation enable the compilation of larger test suites?* To answer this question, we evaluated feasibility of compiling the test code with an increasing number of tests, with and without our transformation.

Within-experiment

- Before treatment: original TC
- After treatment: converted TC

varying sizes, from 10 to 10K tests, for each subject program and measured the compilation times before and after the transformation, enabling all existing compiler optimisations.

the test code? To examine this question, for each program and associated test suite, we compare running times of the original and transformed versions of the test code.

we compared values of internal states and outputs, obtained during execution of each of the tests in the suite with the original and transformed test code.

larger test suites? To answer this question, we evaluated feasibility of compiling the test code with an increasing number of tests, with and without our transformation.

Idea

- propose a CNN-forest method
- Use semantic and structural information
- Locate bugs in source file

Experiment Unit

5 open-source java projects:

Table 1: Subject Projects.

Project	Time Range	# of Bug Reports for Evaluation	# of Bug Reports for Tuning	# of Bug Reports for Training	# of Bug Reports for Testing
AspectJ	03/2002-01/2014	593	100	400	93
Eclipse UI	10/2001-01/2014	3,656	1,500	500	1,656
JDT	10/2001-01/2014	2,632	1,500	500	632
SWT	02/2002-01/2014	2,817	1,500	500	817
Tomcat	07/2002-01/2014	1,056	400	500	156

Treatment

Method:

- combination
- NLP(bug report)
- abstract tree parsing(source code)

Treatment: methods of:

- intrinsic:
  - CNN+forest
  - CNN+CNN
  - forest+forest
- extrinsic:
  - CNN+forest
  - NP+CNN
  - LR+WE
  - DNNLOC
  - BugLocator

Response

MRR and MAP:

- MRR is the mean of the accumulations of the inverse of the ranks of the first correctly-located buggy file for each bug, which is computed in this paper by:

$$MRR = \frac{1}{Q}\sum_{i=1}^{Q}\frac{1}{first_i} \tag{1}$$

where Q is the number of bug reports, $first_i$ represents the ranks of the first correctly-located buggy file for the i_{th} bug report.

- MAP is the mean of average precision values for all Q bug reports. The following is the formulation of MAP used in the field of bug localization:

$$MAP = \frac{1}{Q}\sum_{i=1}^{Q}\sum_{r=1}^{R}\frac{(N_T(r)/r)\times ind(r)}{N_B(i)} \tag{2}$$

where R is the maximum ranks of correct buggy files for i_{th} bug report located by the bug localization technique, $ind(r)$ indicates whether the file located in rank r is correct buggy file ($ind(r) = 1$) or not ($ind(r) = 0$), $N_B(i)$ denotes the number of buggy files for the i_{th} bug report and $N_T(r)$ stands for the number of buggy files correctly located in Top r.

Results

Table 2: Performance Comparison for Intrinsic Evaluation.

Project	Model	MAP	MRR
AspectJ	CNN_Forest	**0.458**	**0.563**
	CNN_CNN	0.449	0.560
	Forest_Forest	0.430	0.549
Eclipse UI	CNN_Forest	0.460	**0.590**
	CNN_CNN	0.441	0.561
	Forest_Forest	**0.465**	0.588
JDT	CNN_Forest	**0.448**	**0.517**
	CNN_CNN	**0.448**	0.502
	Forest_Forest	0.435	0.492
SWT	CNN_Forest	**0.462**	**0.530**
	CNN_CNN	0.415	0.507
	Forest_Forest	0.439	0.512
Tomcat	CNN_Forest	**0.627**	**0.681**
	CNN_CNN	0.623	0.669
	Forest_Forest	0.604	0.647

Table 3: Performance Comparison with the State-of-the-art Techniques.

Project	Model	MAP	MRR
AspectJ	CNN_Forest	**0.436**	**0.519**
	NP-CNN	0.402	0.487
	LR+WE	0.302	0.454
	DNNLOC	0.323	0.475
	BugLocator	0.278	0.364
Eclipse UI	CNN_Forest	**0.432**	**0.534**
	NP-CNN	0.429	0.529
	LR+WE	0.398	0.461
	DNNLOC	0.413	0.514
	BugLocator	0.332	0.383
JDT	CNN_Forest	**0.423**	0.514
	NP-CNN	0.405	0.463
	LR+WE	0.417	**0.516**
	DNNLOC	0.342	0.452
	BugLocator	0.290	0.367
SWT	CNN_Forest	**0.394**	**0.482**
	NP-CNN	0.371	0.466
	LR+WE	0.381	0.446
	DNNLOC	0.369	0.445
	BugLocator	0.269	0.312
Tomcat	CNN_Forest	**0.550**	**0.614**
	NP-CNN	0.529	0.585
	LR+WE	0.503	0.556
	DNNLOC	0.523	0.604
	BugLocator	0.425	0.481

With-in or between?

- **Between-subjects** (or **between-groups**) study design: different people test each condition, so that each person is only exposed to a single user interface.
- **Within-subjects (or repeated-measures)** study design: the same person tests all the conditions (i.e., all the user interfaces).

of conditions to each subject to observe the reactions. The simplest between-group design occurs with two groups; one is generally regarded as the treatment group, which receives the 'special' treatment, (that is, is treated with some variable) and the control group, which receives no variable treatment and is used as a reference (prove that any deviation in results from the treatment group is, indeed, a direct result of the variable.) The between-

Idea

- Is there noise in trace information?

Noise in interaction traces?

- 4 java programs: ECF, jEdit, JHotDraw, PDE
- 15 developers: find bugs/find requirements
- Use a program to record the process
- find differences between manual and recorded information

Experiment Unit

- 4 programs
- 15 people
- Software maintainance on project

Table 3 Systems, tasks used for the experiment and their descriptions

Systems	Tasks	Descriptions
ECF	irc channel output copy/select all works incorrectly	Enter IRC channel (e.g. #eclipse-dev). Right click on the chat output text area to bring up "Copy/Clear/Select All" menu. Choose "Select All". Expected: That channel's text would be selected. Actual: The root container's (irc.freenode.net) output is selected rather than the channel's output.
jEdit	Add lines number and error messages in the select line range dialog	Display next to the "Select line range:" label the number of lines contained in the file and if the user enter a wrong character (everything that's not a number) or a wrong interval you have to display the error message if the user press the "OK" button.
JHotDraw	Change the color of labels and background of the FontChooser application	The "Collections" label will be in blue, the "Family" label in yellow and the "Typeface" label in red. The background of the window will be in black.
PDE	Autostart values are not persisted correctly in the product file	On the Configuration page of the product editor, add a plug-in and set autostart to "true". Save the file. Open the file in a text editor, and see how the value of the "autostart" attribute is still set to false.

Treatment

- Use Mylyn programe to collect data(automatic)
- Use Screenrecorder to analyze data(manual)

Response

- Noise at trace-level
- Noise at event-level

After collecting data, we study noise at both trace and event levels. In fact, noise can be at the trace-level and–or at the event-level: at the trace-level, the noise would impact the overall time spent on the task (i.e., time-related noise) while at the event-level, the noise impacts the characterisation of edit events (i.e., time- and edit-related noise).

”Quasi?”

We recruit participants via emails, which we send to the authors' research groups and individual contacts. We provide potential participants with a link[4] to an online form for registration, collecting information about their level of study, gender, numbers of years of Java and Eclipse experiences. We use this information to assign participants to systems so that participants with different profiles work on a same system to minimise threats to internal validity. To avoid learning bias, each participant perform only one task.

Error Correction

- 3 experiments->case study

64	experiment改为case study	将实例过程中的实验当作了论文使用的方法
71	experiment改为case study	
11	experiment改为case study	将实例过程中的实验当作了论文使用的方法

A Systematic Study on Empirical Software Engineering Research in 2018:Case Study

王瑞华，周康桥，张翔宇

Software Institute, NJU

May 30, 2019

TOC

Contributions

In this presentation:

- We present three different examples of case studies
- We show a standard form of case study, and two with other interesting topics
- We identify and distinguish the difference between the case studies against the standard ones.
- We identify the necessary properties of each case study.

Arrangements

The examples are arranged as follows.

- A case study that is regarded as a standard form.
- A case study with a theoretical replication involved
- A case study which is purely quantitative.

Designing case study

Single case, Embedded Design:

- Data(source) Triangulation: True, Jenkins&Gerrit
- Observer Triangulation: False
- Methodological Triangulation: True, qualitative&quantitative
- Theory Triangulation: True

Preparing data collection

Research Questions:

Table 1 Research questions with description

Research Questions	Objective
RQ1: How and to what extent is Sony Mobile involved in the communities of Jenkins and Gerrit?	To characterize Sony Mobile's involvement and identify potential interviewees.
RQ2: What is the motivation for Sony Mobile to adopt OI?	To explore the transition from a closed innovation process to an OI process.
RQ3: How does Sony Mobile take a decision to make a project or feature open source?	To investigate what factors affect the decision process when determining whether or not Sony Mobile should contribute functionality.
RQ4: What are the innovation outcomes as a result of OI participation?	To explore the vested interest of Sony Mobile as they moved from a closed innovation model to an OI model.
RQ5: How do the requirements engineering and testing processes interplay with the OI adoption?	To investigate the requirements engineering and testing processes and how they deal with the special complexities and challenges involved due to OI.

Experiment Unit: Source

Research Steps(Case study protocol):

1. Preliminary investigation of Jenkins and Gerrit repositories.
2. Mine the identified project repositories.
3. Extract the change log data from the source code repositories.
4. Analyze the change log data (i.e. stakeholders, commits etc).
5. Summarize the findings from the change log data to answer *RQ1*.
6. Prepare and conduct semi-structured interviews to answer *RQ2–RQ5*.
7. Synthesize data.
8. Answer the research questions *RQ1–RQ5*.

Data Collection Source:

From an Organization, About an Organization.

Data Collection

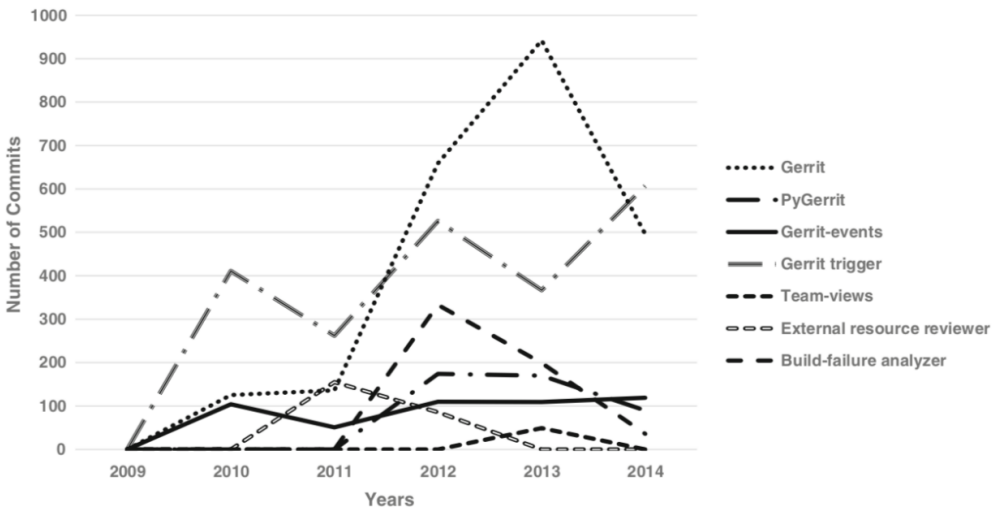
Method: direct method
Data Source:

- archival records
- Interviews

Semi-structured interview:
Three candidates were identified and contacted by e-mail, interviewees 4 and 5 were proposed during the initial three interviews.

Data Analysis

Quantitative Analysis: Basically using Descriptive statistics



Qualitative Analysis: Theory generation/Theory confirmation

1. Transcribe the interviewed data from the five interviewees (see Table 3).
2. Identify and define five distinct themes in the data (see Table 7).
3. Classify the interview statements based on the defined themes.
4. Summarize the findings and answers to the RQs.

Validity: Construct validity/Internal validity/External validity

Reporting case study

Linear-analytic: Standard reporting structure, using a strict step-sequence

Study design

- multiple cases studies
- embedded case studies
- theoretical replications

ABSTRACT

Multitasking has always been an inherent part of software development and is known as the primary source of interruptions due to task switching in software development teams. Developing software involves a mix of analytical and creative work, and requires a significant load on brain functions, such as working memory and decision making. Thus, task switching in the context of software development imposes a cognitive load that causes software developers to lose focus and concentration while working thereby taking a toll on productivity. To investigate the disruptiveness of task switching and interruptions in software development projects, and to understand the reasons for and perceptions of the disruptiveness of task switching we used a mixed-methods approach including a longitudinal data analysis on 4,910 recorded tasks of 17 professional software developers, and a survey of 132 software developers. We found that, compared to task-specific factors (e.g. priority, level, and temporal stage), contextual factors such as interruption type (e.g. self/external), time of day, and task type and context are a more potent determinant of task switching disruptiveness in software development tasks. Furthermore, while most survey respondents believe external interruptions are more disrup-

Research Questions

3.4 Research Questions (RQs)

We formulated the following research questions:

RQ1- Task-specific Vulnerability: How do various interruption characteristics impact the vulnerability of programming, testing, architecture design, UI design, and deployment tasks?

RQ2- Comparative Vulnerability: Which types of development tasks are more vulnerable to task switching/interruptions?

RQ3- Two-way Impact: How does the interaction between various interruption characteristics (ιv_{1-8}) influence the vulnerability of development tasks to interruptions?

Data Collection

- records/survey
- direct/indirect methods

3 METHODS

To achieve our study goals we followed a mixed methods approach including: (1) a longitudinal data analysis on 4,910 recorded tasks of 17 professional software developers, and (2) a user survey with 132 software practitioners to complement the quantitative results with developer perception on task switching and interruptions.

Data Analysis

- Non=parametric kruskal-wallis
- Kruskal-wallis posthoc Test
- p-values
- Spearmans rank(|p|>0.5)
- Phi coefficient tests
- SRH
- modified grounded theory method
- saturate app tool

Table 3: RQ2- Comparison between various development tasks concerning their vulnerability to interruption

Pairs	Context-specific (Dimension 2)								Task-specific (Dimension 1)							
	different context iv_1 (*CS=1*)		different type iv_2 (*TD=1*)		interruption type iv_3 (*IT=1*)		daytime iv_4 (*DT=1*)		priority change iv_5 (*PC=1*)		experience level iv_6 (*EL=0*)		task level iv_7 (*TL=1*)		temporal stage iv_8 (*TS=0*)	
	Δ	$\lvert w \rvert$	Δ	$\lvert w \rvert$	Δ	$\lvert w \rvert$	Δ	$\lvert w \rvert$	Δ	$\lvert w \rvert$	Δ	$\lvert w \rvert$	Δ	$\lvert w \rvert$	Δ	$\lvert w \rvert$
Kruskal-Wallis	3e-4	2e-4	0.001	0.001	0.4	0.05	0.02	0.02	2e-5	4e-6	2e-4	0.003	4e-4	1e-4	0.1	0.03
Programming-Architecture	0.02*	0.03*	0.2	0.2	0.2	0.04	0.03*	0.1	0.004*	0.003*	0.8	0.04*	0.01	0.05	0.3	0.2
Programming-Test	0.001	0.001	0.01	0.1	0.4	0.9	0.6	0.4	0.003	0.3	0.3	0.4	0.3	0.25	0.07	0.2
Programming-UI	0.5	0.03*	0.2	0.05	0.3	0.001	0.04	0.02	0.4	0.2	0.04	0.01	0.04	0.02	0.1	0.08
Programming-Deployment	0.1	0.06	0.1	0.01	0.3	0.2	0.9	0.05	0.02	0.01	0.01	0.01	0.01*	0.02*	0.1	0.01
Test-Architecture	0.05*	0.03*	0.9	0.8	0.1	0.04	0.07	0.3	0.3	0.6	0.7	0.4	0.05	0.3	0.9	0.6
Test-UI	0.2	0.04*	0.9	0.02*	0.2	0.01	0.5	0.9	0.2	0.1	0.01	0.01	0.2	0.2	0.4	0.2
Test-Deployment	0.2	0.001	0.01	0.002	0.5	0.2	0.02	0.02	0.1	0.03	0.1	0.04	0.001*	0.002*	0.02	0.001
Architecture-UI	0.9	0.9	0.9	0.7	0.9	0.5	0.07	0.3	0.07	0.1	0.1	0.2	0.4	0.8	0.6	0.5
Architecture-Deployment	0.08	0.02	0.04	0.001	0.2	0.02	0.1	0.01	0.4	0.04*	0.1	0.02	0.2	0.02	0.05	0.003
Deployment-UI	0.1	0.06	0.3	0.01	0.2	0.01	0.001	0.002	0.02	0.01	0.001	1e-4	0.1	0.6	0.02	0.001

**: The p-value of the alternative value of the corresponding variable.*

Results

- Contextual factors disruptive

- Deployment tasks seem more vulnerable

4.1 RQ1- Task-specific Vulnerability

We follow a template and posed 80 null hypotheses to explore factors that may explain the disruptiveness of interruptions in various types of software development tasks: H_0 = **Interruption characteristic** ιv_i **does not impact the** Δ **and/or** $|w|$ **of task switchings in task** $\langle T \rangle$**.** Where Δ denotes the suspension period, $|w|$ the length of nested task switching, and $\langle T \rangle$ denotes the task type. As illustrated in Table 2, of 34 (43%) rejected tests, 21 (62%) are related to *contextual* factors, and 13 (38%) are related to *task-specific* factors. This implies that, compared to task-specific factors, contextual factors (e.g. context switching and interruption type) are more potent determinants of task switching disruptiveness in software development tasks.

4.2 RQ2- Comparative Vulnerability

We posed 160 null hypotheses following this template: H_0 = **The disruptive impact of** ιv_i **on** Δ **and/or** $|w|$ **is not different between tasks** $\langle T, T' \rangle$, where $i \in \{1, 2, \ldots, 8\}$ and ιv_i, and $\Delta/|w|$ denote independent variables and disruptive factors, respectively. T and T' represent two different task types for all possible pairs of task types (i.e. $\binom{5}{2} = 10$ pairs). Table 3 presents the p-value for each of these tests. The results of our 95% confidence interval analysis (e.g. Figure 6a-q) show that in all cases that task or context-specific factors make a significant difference between deployment and other development tasks, deployment tasks are more vulnerable to interruptions than other task types. This could be because deployment tasks are highly interdependent on different tasks within a development process, which makes their resumption more complicated due to the associated tasks.

Conclusion & Validation

- Threats solved:
  - Different levels and various countries of survey objects
  - Various projects, different business domains, different levels of exp.
  - Cohen's Kappa statistic: Kappa value=0.87
- Proposition
  - Measure and characterizing the cost of task switching and interruptions
  - Can be a productive task switching

Background

- WIP: work-in-process
- kanban
- lead time
- productivity

Researh Question

- RQ1: Which WIP optimizes team performance?
- RQ2: What are the challenges of quantifying the effect of WIP limit on the performance of Kanban teams?

Case Study Design

- simple case study(one company)
- embedded case study (unit of analysis: lead time, productivity)
- not a replication

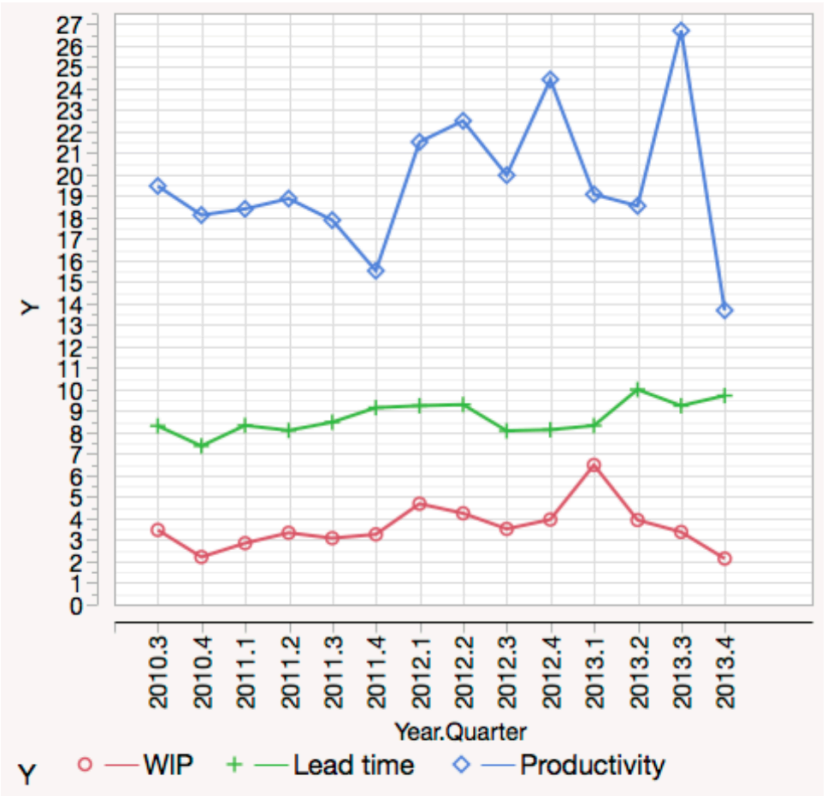
Data Collection

physical artifacts:

- 8,505 work items
- by five teams
- over 3.5 years(2010 second half - 2013 end)

Results

- an increasing WIP: 2.8(2010)3.1(2011)4.1(2012)4.0(2013)
- with a longer lead time: 7.8 days(2010)9.3 days(2013)
- high correlation between productivity and WIP: statistically significant($\rho = 0.71, \quad p < 0.01$)



Validity

- Construct validity: code churn to measure work item size?
- Internal validity: variation in work item size is controlled?
- External validity: other companies?

Data was generated before the study was designed.

Corrections

ESEM40:Software Quality Assessment in Practice: A Hypothesis-driven Framework

- This paper proposed a structured framework to link abstract model and data.
- The method is NOT an automatic tool.
- Requires human interaction.
- Investigated properties of two systems.
- Wrong category: not a case study

A Systematic Study on Empirical Software Engineering Research in 2018:Survey

陈俊达，雷诚，张傲

Software Institute, NJU

June 18, 2019

TOC

Overview

- 2 Roles of survey
  - Main:As main approach for data collection
  - Part:As part of case study process
- 2 Forms of survey
  - Interview
  - Questionaire
- 3 Purposes
  - Evaluate: What's the rate/frequencey of certain characteristics related to research question?
  - Assess: What's the severity of certain caracteristics or conditions?
  - Identify: Which factors are related to certain caracteristics or conditions?

Arrangements

Papers are arranged as follows:

- ESEM 28：Main, Questionaire, Evaluate.
- EMSE 105: Main, Interview&Questionaire, Assess.
- EMSE 31: Part, Interview, Identify.
- We distinguish them by combining three factors of surveys.

ESEM28

- Understanding the Software Development Practices of Blockchain Projects
- Online survey
  - Towards blockchain software (BCS) developers on GitHub
- Explore the software engineering practices adopted in BCS
- The first formal survey

tices. **Aims:** To bridge this gap, we aim to carry out the first formal survey to explore the software engineering practices including requirement analysis, task assignment, testing, and verification of blockchain software projects. **Method:** We sent an online survey to 1,604 active BCS developers identified via mining the Github repositories of 145 popular BCS projects. The survey received 156 responses that met our criteria for analysis. **Results:** We found

Design&Develop

- Cross-sectional
- Both open and closed questions
- Including demographic questions

Table 1: Survey Questions

#	RQ*	Question Text	Answer Choices
Q1	D	How many years of software development experiences do you have?	[Less than a year, between one to five years, between six to ten years, more than ten years]
Q2	D	How many years have you been developing blockchain software?	[Less than a year, between one to two years, between three to five years, more than five years]
Q3	D	What is your primary Blockchain software project (i.e. the project that you have spent most of your time)?	[#]
Q4	D	What are your roles in your primary project (please check all appropriate roles)?	[Developer, Maintainer, requirement analysis, Testing, User support, Documentation, Social marketing]
Q5	D	Approximately, how many pull requests have you submitted to your primary project?	[Less than 10, Between 11 to 30, More than 30]
Q6	D	Approximately, how many hours on average do you spend per week on your primary project?	[Less than 5, between 6 to 10, Between 11 to 20, Between 21 to 35, I work full time]
		Please answer the following questions based on your own experiences with your primary project (i.e., as responded in Q3.)	
Q7	RQ6	Which of the following channels do you use to communicate with the peers from your primary project? Please check all that apply.	[email, Slack, Discord, Github, instant messenger, Skype, mailing list, Reddit, Medium, others (please specify)]
Q8	RQ1	Which of the following software development practices do you follow (Please check all that apply)?	[peer code review, automated build (aka one-click build using ANT, Maven, Gradle, CMake), continuous integration (e.g., Travis, Jenkins, Teamcity, Bamboo, Buildbot, or Cruisecontrol), pair programming, unit testing, automated testing, test driven development (aka test first development), performance testing, formal verification, others (please specify)]
Q9	RQ1	Please rank the following software development practices based on their effectiveness to improve the quality of your project (you can drag and drop to reorder the following list):	[formal verification, automated testing, pair programming, performance testing, test driven development (aka test first development), unit testing, continuous integration (e.g., Travis, Jenkins, Teamcity, Bamboo, Buildbot, or Cruisecontrol), peer code review, automated build(aka one-click build using ANT, Maven, Gradle, CMake)]
Q10	RQ4	How the requirements of your projects are identified and selected?	[#]
Q11	RQ5	How development tasks are assigned among the project members?	[#]
Q12	RQ2	How do you verify the correctness of your code?	[#]
Q13	RQ3	How do you test your software for security and scalability?	[#]

**numbers refer to the research question that motivated the inclusion of the survey question, 'D' refers to demographic questions*

Evaluate

- Pilot survey
  - CS Prof. & graduates
  - Minor changes
- Focus groups
- Face validity

4.3 Pilot Survey

To help ensure the understandability of the survey, we asked Computer Science professors and graduate students with experience in SE and experience in survey design to review the survey to ensure the questions were clear and complete. The feedback only suggested minor edits. The changes we made include: adding more

Obtain Data

- Non-probabilistic sampling
  - Convenience sampling
  - Send to all qualified developers and get responses from whom was available at the time
- Sample Size: 156
- Response rate: 156/1604

4.2 Participant Selection

To ensure valid results, we only surveyed BCS developers with sufficient experience. We identified 145 BCS projects based on following four criteria:

- Tagged under at least one of the following six 'topics'[3]: `blockchain`, `cryptocurrency`, `altcoin`, `ethereum`, `bitcoin`, and `smart-contracts`.
- 'Starred' by at least ten users.
- Have at least five distinct contributors.
- A manual verification of the repository confirmed it as a BCS project.

We used Github API[4] to identify 1,604 contributors, each of whom had submitted at least five changes to one of those 145 projects. We mine the Git commit logs of the identified 145 projects to gather the email addresses of those 1,604 active contributors.

Data Analysis

For open questions.
A systematic qualitative data analysis process:

- **Extract** general themes
- **Reach** agreed-upon coding scheme
- **Code** the responses with CAT
- **Calculate inter-rater reliability**
- **Resolve** discrepancies
- **Transfer** data to SPSS

4.5 Qualitative Analysis Process

For the open-ended questions, we followed a systematic qualitative data analysis process. First, two of the authors independently extracted the general themes from the first 75 responses to each question. Using those themes, the authors had discussion sessions to develop an agreed-upon coding scheme for each question. Using this coding scheme, another author went through the remaining answers to determine any additional codes that need to be added.

With this scheme, two of the authors independently coded each response using the Coding Analysis Toolkit (CAT) [12] software. The coders could also add new codes, if necessary. We computed the level of inter-rater reliability of the manual coding process using Cohen's kappa [4], which was measured as 0.62. While there is no universally accepted 'good' kappa, values between 0.61 to 0.80 are generally recognized as 'substantial agreements' [9]. We used CAT to identify the discrepancies in coding and had discussion sessions to resolve all conflicts. Once we completed the coding process, we transferred the data into IBM SPSS for further analysis along with the quantitative data.

Inter-observer (inter-rater)

– the reliability of surveys that involve a (group) of trained person completing a survey instrument based on their own observations

Result

- 6 RQs
- Data Visualization
- Data Description
- Analyze causes

RQ1: *Which are the software development practices that BCS developers follow?*

RQ2: *How do BCS developers verify the correctness of their software?*

RQ3: *How do BCS developers test their software for security and scalability?*

RQ4: *How do BCS developers identify and select the requirements for their projects?*

RQ5: *What are the task assignment procedures among the BCS projects?*

RQ6: *What are the communication channels for the BCS developers?*

Result

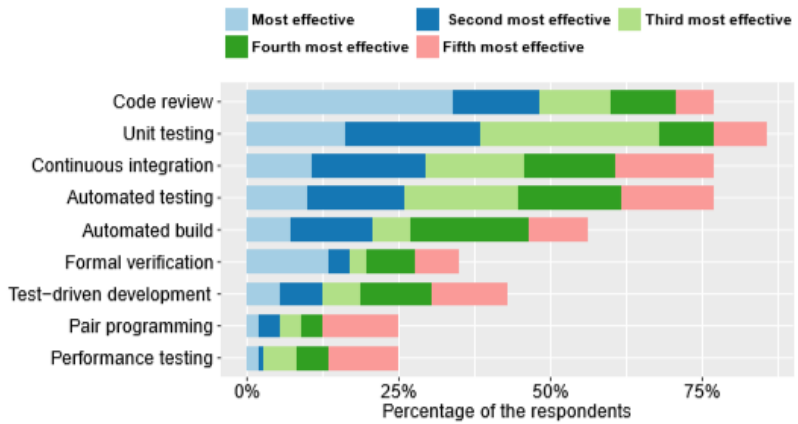


Figure 5: Ranks of software development practices based on effectiveness

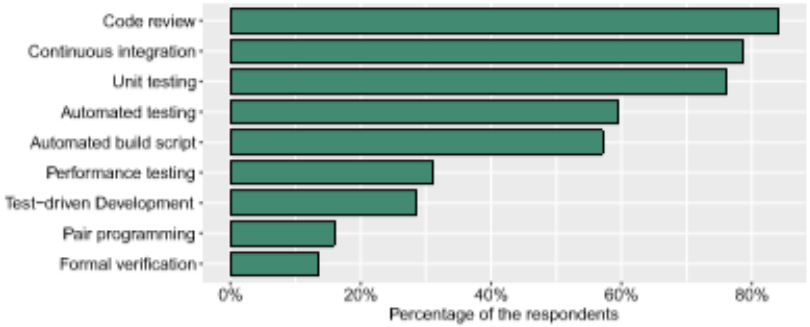


Figure 4: SE practices of BCS developers

6.1 RQ1: Software Development Practices

Figure 4 shows different software engineering practices of BCS developers and Figure 5 ranks them based on their perceived effectiveness, which emerged from the answers to Q8 and Q9 (Table 1) of our survey.

It is evident from Figure 4 that code review is the most common practice. Continous integration and unit testing are also widely used by BCS developers. Figure 5 shows that according to the BCS developers code review is also the most effective one.

As a young and immature ecosystem, the BCS domain lacks adequate testing tools and without that humans are the best bug finders in BCS projects. Although formal verification ranks sixth in terms of ef-

Objective&Method

Objective: to assess the severity of some characteristic or condition that occurs in a population.

delays for rapidly releasing projects—98% of the fixed issues in the rapidly releasing Firefox project had their integration delayed by at least one release. To better understand the impact that rapid release cycles have on the integration delay of fixed issues, we perform

Method: Qualitative

1.2 Qualitative Study

Next, we survey 37 developers from

Design

Research design: This study not only has a survey, but also has an interview.

> In Study II, we qualitatively analyze the integration delay phenomena by surveying and interviewing the team members of our subject projects.

Survey design: Cross-sectional

> years. To collect the data, we designed a web-based survey that was sent to 780 developers of the Firefox, Eclipse (JDT), and ArgoUML projects. We sent our survey to 513 Firefox, 184 Eclipse (JDT), and 62 ArgoUML developers whose e-mails existed on the mailing lists.

Survey Details

- Self-administrated questionnaires: web-based
- Interviews: semi-structured, assumed to be one-to-one
- Does not use existing instruments

Sampling

- non-probabilistic sampling, web-based survey
- convenience sampling
- 37 responses received, 5% rate.

Non-probabilistic sampling

– respondents are chosen because of easy access or researchers' justification for their representativeness of the population

– **convenience** sampling: obtaining responses from people who are available and willing to take part

Data Analysis

Participant experiences

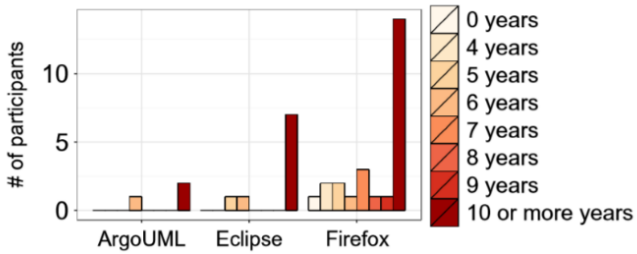


Fig. 5 Software development experience of the participants

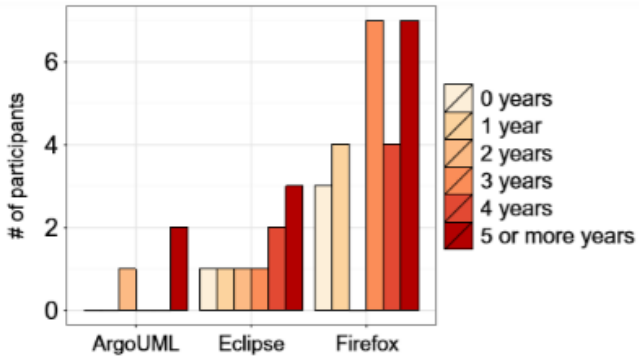
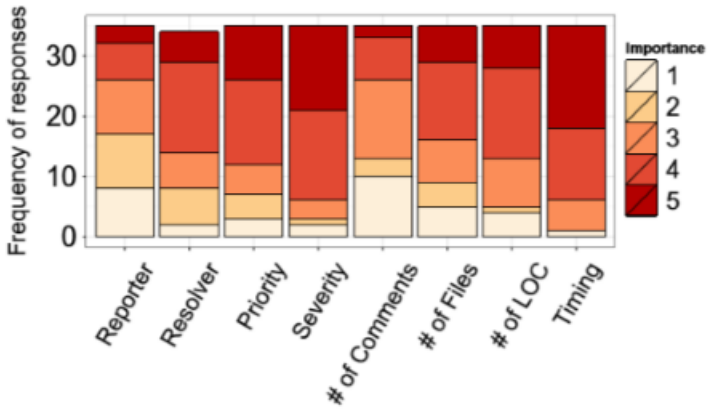


Fig. 6 Development experience of the participants in the respective project

Fig. 17 Frequency of ranks per factor



To what extent do you agree that the characteristics listed in the table below are related to the delivery delay of a completed issue?

Mark only one oval per row.

	Strongly agree	Agree	Neither agree nor disagree	Disagree	Strongly disagree
The reporter of a completed issue					
The resolver of a completed issue					
The priority value of a completed issue					
The severity value of a completed issue					
Number of comments recorded in a completed issue					
Number of files modified to complete an issue					
Number of lines of code to complete an issue					
The time at which an issue is completed during a release cycle					

Introduction

Motivation:

- Previous works: mainly on tracing changes of low-level project features, e.g. bug fixes.
- This paper: on the change of high-level feature:requirement.

Contribution:

- Show an emperical study of whether requirements can be finished in planned iteration.
- build a predictive model to predict whether requirements can be finished in planned iteration.
- All the work was based on three IBM project

Survey Administration

- Interview: face to face
  - 16 semi-structured 30-60 minute interview
- Interviewee:
  - IBM program director
  - Analytics architect
  - 3 other IBM practitioners

Roles of Interview

- Drive the study design
- Validate understanding of the repository data
- Find factors which matter in building a prediction model
  - Prediction model features: features proposed in previous works + new features through interviewing

Roles of Interview-exempli gratia

3.1.9 Stakeholder Characteristics

Through the interviews with IBM practitioners, we learned that the types of stakeholders involved in a work item can impact its completion time. To account for this, we consider:

Springer

Empirical Software Engineering

- DAYS_SINCE_LAST_DE_COMMENT: number of days since a distinguished engineer (DE) commented on the work item. In IBM, DE is a title reserved for very respected developers. If a DE participates in the discussion of a work item, it will likely receive prompt attention.
- COMPONENT_RESOLVER: the total number of work items that the work item owner has resolved in the same component up until the modified date of the corresponding history of each requirement. This indicates the expertise of the owner in the domain of the problem.
- CREATOR_TEAM_RELATIONSHIP (Guo et al. 2010; Marks et al. 2011): the relationship of the work item creator to the assigned team. Prior work suggests that bugs reported by people on the same team are likely to get fixed faster (Guo et al. 2010). For this feature, the creator could be an IBM developer from the same team, an IBM developer from another team, or a customer from outside the company.

Purpose&Design

Objective: find factors

Design:

- longitudinal: Multiple iterations to find out related factors
- Involves different people, do not emphasize differences.

Sampling

- No probability methods
- Uses convinience sampling
- Mainly rely on interviewee's experience rather than data analysis

Analyzing

- Directly uses the data.
- Includes:
  - proposals of previous works
  - suggestions of IBM interviewers

Corrections

- 2 papers removed: ESEM22&ESEM37
- Reason: did not actively **ask** questions, but merely analyze **exsisting** data.

An empirical study of ARDOC Systems: Empirical Engineering Final Report

Yuqing Yang, Cheng Lei, Kangqiao Zhou

Software Institute, Nanjing University

June 27, 2019

TOC

ARDOC system

- Panichella et al., App Reviews Development Oriented Classifier, ACM SIGSOFT'16
- Palomba et al., CHANGEADVISOR, ICSE'17
- Providing filetering and clustering for developers to comprehend review
- Uses HDP(Hierarchical Dirichlet Process) to categorize
- Judge by c_v coherence

Research Questions

- RQ1. How well is the clustering results of the ARDOC-like process?
- RQ2. Does the ARDOC-like process benefits both development efficiency and developing experience?

Selecting appropriate methods

- Case Study: least possible.
- Experiment: promising.
- Survey: not much, but nice to intergrate questionaire
- HE SAID YES

Selecting appropriate methods

- Case Study: least possible.
- Experiment: promising.
- Survey: not much, but nice to intergrate questionaire
- HE SAID YES

Selecting appropriate methods

- Case Study: least possible.
- Experiment: promising.
- Survey: not much, but nice to intergrate questionaire
- HE SAID YES

Selecting appropriate methods

- Case Study: least possible.
- Experiment: promising.
- Survey: not much, but nice to intergrate questionaire
- HE SAID YES

Research Protocol

- Model training. train the model under the process defined in CHANGEADVISOR judging by c_v coherence.
- Pre-Assessment. Twenty actual users invited, whether the results fits the user demands of the system.
- Development. Between-subject design. Sixteen developers are invited in our experiment on a task.
- In-Assessment. Five experienced project developers invited as experts to assess the time and quality.
- Post-assessment. Each new-commer developers: do a questionaire commenting on using experiences.

Potential Threats

- User Personal Bias: non-professional users' bias
- Before-after Bias: between, not with-in
- Developer Selection Bias: variations on developer backgrounds
- Evaluation Bias: first-impression effect

Model Training

We train the model according to the description in CHANGEADVISOR. Then we evaluate the performances of each run, calculating the coherence. Empirically, we expect the ideal coherence to be around 0.7-0.75. After the model is trained, we push our experiment to the next phase.

- train model in CHANGEADVISOR
- choose best parameter for highest coherence
- 0.7-0.75 is ideal

Pre-Assessment

- twenty actual users invited
- 500 random-selected comments

Development

- 16 developers invited
- fill in a questionaire about working years etc
- balanced into two groups
- do a task on problem discovery

Questionaire Design:

1. How long have you been programming?
2. Your familiar programming language?
3. What's your eduacation status?
4. What's your current job title?

In-Assessment

- five developers of the app invited(expert), plus users
- evaluate quality

1. Experts: Code quality, 1-10
2. Users: Functional quality, 1-10

Post-Assessment

- developers fill in a questionaire about experience

Questionaire design:

1. Clustering accuracy
2. Code Correlation

Expected Result

Item	Exp.(with)	Compar.(without)
Homongeneity of clustering	high	N/A
Development time	fast	slow
Development quality	high	low
Developer experience	good	fair

Table 1: Expected result