Fall 11-5-2015

Candoia: A Platform and an Ecosystem for Building and Deploying Versatile Mining Software Repositories Tools

Nitin M. Tiwari
Iowa State University, nmtiwari@iastate.edu

Dalton D. Mills
Iowa State University, dmills@iastate.edu

Ganesha Upadhyaya
Iowa State University, ganeshau@iastate.edu

Eric Lin
Iowa State University, eylin@iastate.edu

Hridesh Rajan
Iowa State University, hridesh@iastate.edu

Candoia: A Platform and an Ecosystem for Building and Deploying Versatile Mining Software Repositories Tools

Software Repository Mining? There is an App for That!

Nitin M Tiwari Dalton D. Mills Ganesha Upadhyaya Eric Lin Hridesh Rajan
Iowa State University
Ames, IA
{nmtiwari,dmills,ganeshau,eylin,hridesh}@iastate.edu

ABSTRACT

Research on mining software repositories (MSR) has shown great promise during the last decade in solving many challenging software engineering problems. There exists, however, a 'valley of death' between these significant innovations in the MSR research and their deployment in practice. The significant cost of converting a prototype to software; need to provide support for a wide variety of tools and technologies e.g. CVS, SVN, Git, Bugzilla, Jira, Issues, etc, to improve applicability; and the high cost of customizing tools to practitioner-specific settings are some key hurdles in transition to practice. We describe *Candoia*, a platform and an ecosystem that is aimed at bridging this valley of death between innovations in MSR research and their deployment in practice. We have implemented Candoia and provide facilities to build and publish MSR ideas as *Candoia apps*. Our evaluation demonstrates that Candoia drastically reduces the cost of converting an idea to an app, thus reducing the barrier to transitioning research findings into practice. We also see versatility, in Candoia app's ability to work with a variety of tools and technologies that the platform supports. Finally, we find that customizing Candoia app to fit project-specific needs is often well within the grasp of developers.

Categories and Subject Descriptors

[**Software and its engineering**]: Software evolution, Application specific development environments, Software repositories

Keywords

Analysis of software and its evolution, MSR, research to practice

1. INTRODUCTION

Over the last decade, mining and understanding software repositories (MSR) research has shown significant advances in several critical software engineering (SE) areas. Key successful research sub-areas in MSR are: defect prediction using a) product metrics [23, 55, 7, 14, 76, 41, 74], b) process metrics such as code changes [59, 34, 43, 62, 36, 72], c) source code dependencies [90, 60], d) socio-technical network, organization structure, and human factors [11, 17, 58, 61, 66, 54, 86, 84, 5, 12, 45], etc.; fixing effort estimation and suggesting fixing experts [20, 4, 10, 40, 48, 67, 52, 16, 15, 53, 8, 70]; social network analysis for software development [11, 17, 58, 61, 66, 54, 86, 84, 5, 12]; programming pattern discovery [63, 82, 50, 26, 1, 85, 18, 79, 38, 49, 47, 46, 42, 78, 56, 3, 75, 68]; specification mining [89, 81, 27, 88, 71, 51, 2, 37, 49, 22, 21, 24, 83], etc. However, this research has not yet focussed on the question of widely-distributing MSR tools. There are frameworks and platforms that aim to ease deployment of program analysis tools [80, 73]; however, their focus is not on MSR.

There are at least three major technical challenges in transitioning MSR research to practice. First, most researchers, and adequately so, focus on realizing their research as a program suitable for their own experiments — there is a significant cost to converting these prototypes into software intended for wide usage. Developing MSR tools as plugins for platforms like Eclipse can help with that, however, the level of abstraction that these platforms provide is still too low for MSR tools. Second, in order to be broadly applicable a MSR research prototype must integrate with a variety of tools — version control systems (VCS) such as CVS, SVN, Git, etc., bug databases such as Bugzilla, Issues, Jira, etc., forges such as SF.net, GitHub, Bitbucket, etc., programming language such as Java, Javascript, PhP, etc. Expecting such broad applicability from every tool meant to evaluate research hypotheses may not be reasonable, and may substantially increase the cost of scientific research in this area. Third, usage scenarios that the researchers may have used in their experiments may not exactly match the need for all users of their tool, requiring users to slightly customize the tools to fit their purpose. If the implementation of the tool is too complex that may challenge the resolve of users to customize the tool for their own projects. These three challenges substantially increase the cost of research-to-practice transition in this area.

To solve these problems, we have designed a platform for realizing MSR tools that we call Candoia (pronounced can-doy-uh). Candoia is to MSR tools as Android and iOS are to mobile applications, and Atom[1] is to text editors. Like these platforms, Candoia provides suitable abstractions for building MSR tools and handles the details of integration with VCS, bug databases, language parsers, visualization, etc. Researchers prototype their research as *Candoia apps* and can publish them. Practitioners install Candoia platform and can browse through available apps, download them, and if necessary customize them for their own needs. We have implemented Candoia and make it available for download[2]. We have also created an exchange for Candoia apps, and populated it with an initial set of apps. The Candoia platform can interact with the exchange to fetch published apps.

[1]Atom, a hackable text editor available at: http://atom.io

[2]Candoia, an ecosystem for MSR tools: http://candoia.org

ICSE '15 Austin, Texas USA

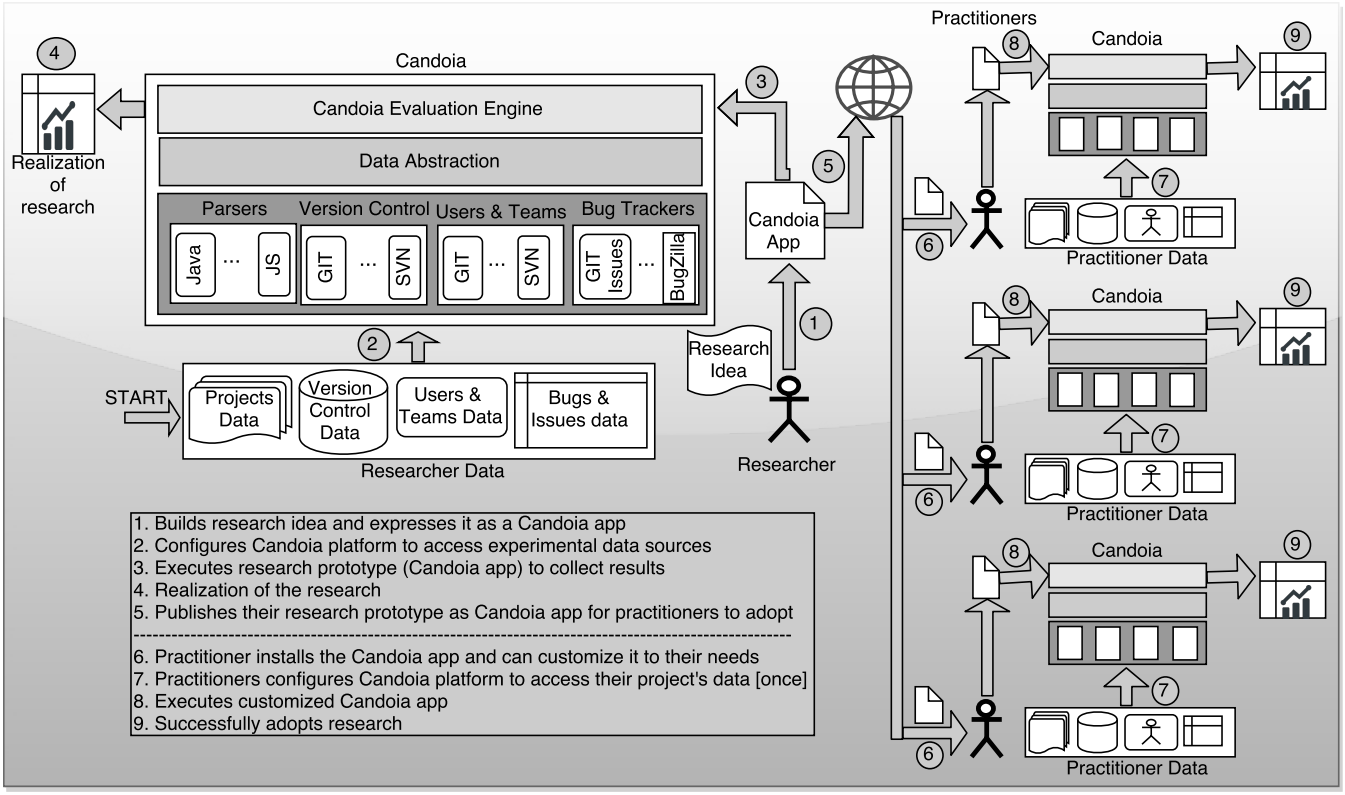

Figure 1: Candoia Operational Overview

To evaluate Candoia's design and effectiveness we have created over two dozen apps inspired from MSR research in four different areas: (1) bugs and defects, (2) software evolution, (3) teams and project management, and (4) programming practices. Our results show that Candoia substantially decreases the efforts required for a research result prototyped in one experimental setting to be applied to practice in substantially different project settings. Candoia apps scale well to such large projects as Hadoop, JQuery, and SLF4j. Last but not least, we find that customizations of provided apps, to suit a practioner's purpose better, are easy to make.

We now describe Candoia and explore its advantages. First we present the Candoia platform in §3. §4 presents our studies of applicability, adoptability, customizability, interoperability, and scalability. §5 describes related work and §6 concludes.

2. MOTIVATION

Today we have a 'valley of death' between significant innovations in the mining software repositories research and their deployment in practice If these innovations can be made more accessible to practioners, they can contribute to overall improvement in software development practices and lead to higher software quality. To illustrate the problem, consider a research team that has a research idea for recommending *file changes based on change history*. Typically the research team would create an experiment setup including a dataset for testing their idea. Say for instance, they setup an experimental dataset consisting of several open source Java projects from SourceForge[3]. The research team would then build an experimental infrastructure for reading SVN repositories, Java source code, and organization data from SourceForge for these projects.

[3]SourceForge is a repository for open source projects (http://sf.net).

Imagine that the research team has successfully verified their hypothesis, published their research results, and is now interested in broad applicability of their ideas in practice. This process today is very intimidating. The team would need to enhance their experimental infrastructure to include support for common version control systems (VCS), e.g. Git, CVS, etc. to allow practioners using these VCS to use the research result. If the team is interested in making their research results available to users of other languages, those languages will need to be supported. Visualization support would be needed to communicate results. Last but not least, to reach users of different repositories e.g. GitHub, added support for these repositories will also be desirable. These needs together significantly increase the cost of transitioning research to practice.

There is a similar challenge at the other end — adopting research in practice. Imagine that a brave research team has invested efforts in developing their research tools with broad applicability in mind, and make it available as an open source tool. The resulting MSR tool's complexity will invariably match its feature richness. However, each practioner's use case can differ slightly from that anticipated and implemented by the tool, e.g. data sources may be named and organized differently (e.g. the name of the master branch in SVN), threshold values may be different (e.g. searching for top-2 as opposed to top-5 Javascript developers for a critical project deliverable), the project might call for a different visualization of results (e.g. bar vs. line chart), the project might need a different set of keywords (e.g. fixed vs. similar meaning word in spanish to search for fixing revisions), etc. To accomodate these needs, a tool can provide configuration files that the practioner can edit and read these configuration values into its implementation, but building that support further increases the complexity and the time investment needed from research team.

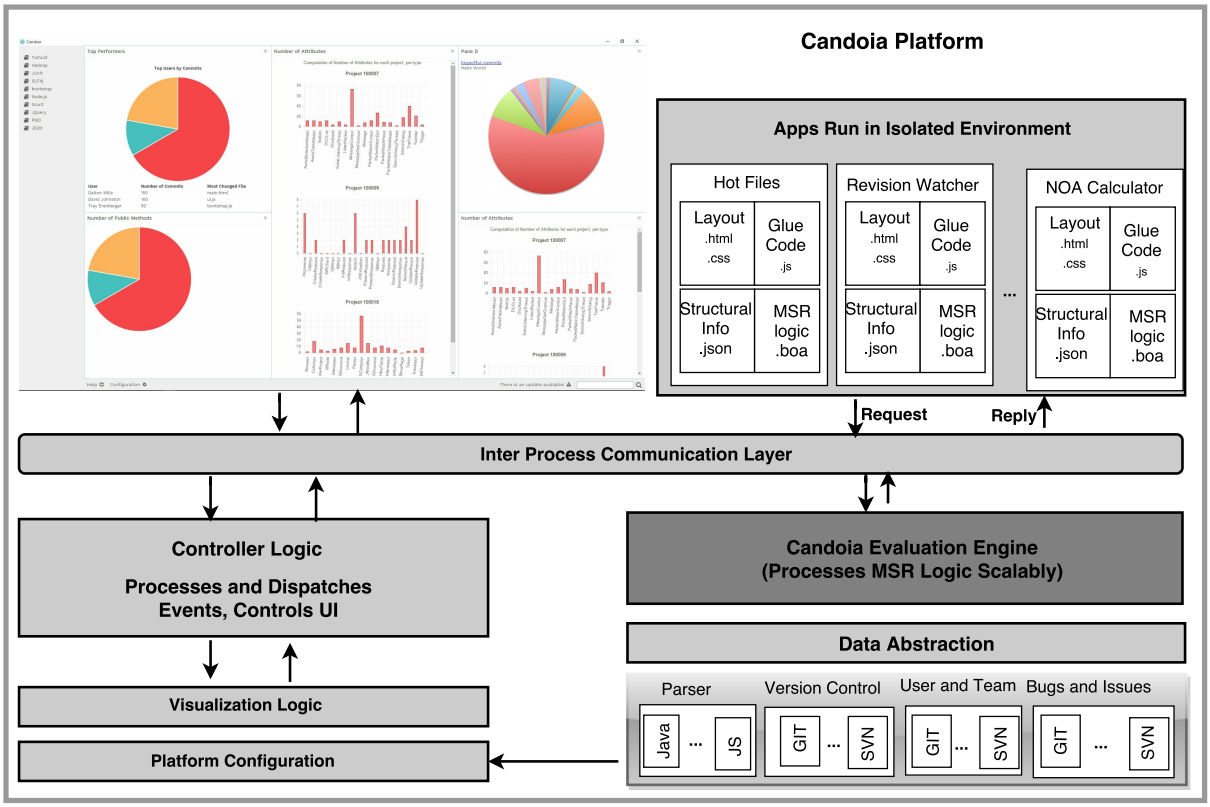

Figure 2: An Overview of Candoia Platform's Architecture

3. CANDOIA PLATFORM & ECOSYSTEM

We have designed and implemented a platform that we call *Candoia* to solve these problems. Candoia is aimed at significantly decreasing the cost of transitioning MSR research to practice. The main features of Candoia are inspired from existing successful platforms and ecosystems, namely the Android and the iOS platform, but bring similar benefits to fruition for MSR tools. Figure 1 shows the operational overview of the Candoia platform and ecosystem and Figure 2 shows the detailed architecture of the platform.

To illustrate Candoia's properties, let us reconsider our research team that has a research idea for recommending file changes based on change history. The research team realizes their ideas as a Candoia app. A Candoia app consists of four parts: the MSR logic, the structure description of the app, the layout description for the visualization, and glue code. Each of these parts are relatively short because they are written using languages designed specially for that purpose (more on that later in this section). The research team can configure their own Candoia platform to add URLs for SourceForge projects that they wish to include in their experimental dataset. The platform then downloads and creates a cached version of selected projects for the experiments[4]. The research team can also configure the preferences of the platform to select their local projects. The research team can then run their experiments and visualize results. So, if desirable, a research team can perform both their research experiments within the Candoia platform and use it as a vehicle for dissemination and increasing broader impacts. Alternatively, they can also first focus on the MSR logic and defer work on other parts of a Candoia app until later. The MSR logic code focuses only on essential parts of the research idea because support for other commonly used functionalities such as iterating over revisions, traversing abstract syntax trees, etc., is provided by the Candoia platform.

After successfully validating their hypothesis, if the research team is interested in broadly disseminating their research results they can upload their Candoia app with the prespecified structure in a Git repository[5]. Conforming to the prespecified app structure is currently required by the platform. Since the platform includes supports for common VCS, languages[6], visualization, and remote repository access these hurdles for research-to-practice transition are removed. The research team can then list the URL of their app's Git repository on Candoia's exchange[7] to publish their app and disseminate their research tool. This is as simple as filling out a single-page short form.

A practioner can download and install the Candoia platform on their computer system, and configure their platform to point it to their project's repository. This is as simple as pasting a local directory name or remote URL. When the Candoia platform is configured with a new repository, it caches information from the version control system, bug repository, and other project-related metadata (if such information is available).

The practioner can also browse and add Candoia apps to their platform. These apps can then process any project that is available within the platform. This removes the need to customize each MSR

[4]Downloading and running apps on remotely hosted projects is available for both SourceForge and GitHub as of this writing.

[5]Currently works with GitHub (http://github.com), but support for other repositories is certainly possible.

[6]Java, Javascript, and PhP at the time of this writing.

[7]Currently Candoia's exchange provides a categorized listing of apps. In future, it can include features such as ratings, comments, etc, but for now we have focused on functional components.

tool to point to project-specific data source, rather it is done once for all apps. If necessary, a practioner can also view the structure and parts of the app and edit them[8]. Since these parts have smaller number of lines of code (because majority of the concerns are implemented and provided by the platform), practioners may be more successful in customizing them. Indeed, we provide some preliminary evidence in §4 that shows ease of customizability. Another important benefit is that a practioner can inspect the code for MSR logic to ascertain if the app satisfies their intended purpose. These benefits make the Candoia platform a positive step forward toward bridging the research-to-practice gap for MSR results.

Technical Challenges.

There were several technical challenges that had to be overcome to realize the overarching research goals of the Candoia platform.

1. How to make it easier to describe various functionalities of a Candoia app, especially MSR logic and visualization?
2. How to facilitate ease of customization of Candoia apps by practioners?
3. How to secure a practioner's system against third-party Candoia apps, and secure one app from other apps?
4. How to enable cross platform deployment of Candoia with familiar look-and-feel across platforms?
5. How to facilitate integration of a large number of disparate data sources and sinks within the Candoia platform?

Our key insight to address the first two technical challenges was to select a reasonably well-known domain-specific language for expressing each functionality — with significant number of abstractions for easily expressing that functionality. For instance, for visualization and layout of Candoia apps we selected the well-known combination of HTML and CSS, for describing the structure of Candoia apps we selected JSON, for describing the MSR logic we selected Boa [25], and for writing glue code to manage interaction, updates, and data exchange in an app we selected Javascript.

Structure description of a Candoia app.

The structure description of a Candoia app is described by its `.json` configuration file. An example structure description appears below:

```
{
  "name": "example-app",
  "version": "0.1.0",
  "author": "MSR Guru <msrATguru.com>",
  "description": "An example Candoia app",
  "main": "main.html",
  "homepage": "https://github.com/user",
  "repository": {
    "type": "git",
    "url": "https://github.com/user/example.git"
  },
  "keywords": [
    "example",
    "candoia",
    "application",
  ],
  "icon": ["app.ico", "app.png", "app.gif"]
}
```

Some of these fields are required (for example, the `"main"` must point to a valid `.html` file within the repository). The rest of the structure can be application-specific. (including organization to it's own resources/files, and a `key/value` store on the user machine.) The Candoia website also uses the .json file to auto-fill fields that describe the app.

The layout and visual appearance of an app is described using HTML and CSS. Within an app's HTML code, the app developer is able to add any Javascript code or link to any Javascript or CSS files they want (including 3rd party libraries).

Security Architecture of the Candoia platform.

A key concern for Candoia is to allow apps to communicate with the Candoia platform in a safe way, and to allow access to practioner's data on a need-to-know basis. We also need to prevent apps from corrupting each other.

We have solved this technical challenge (#3) as well as our cross platform deployment challenge (#4) by building on top of the Chromium platform [77]. Chromium is an open source, cross platform browser. Candoia builds on the process architecture of Chromium, where each window runs in an isolated process. In Candoia each app runs in its isolated process, and it can communicate with a special process that we call *controller process* via inter-process communication (ipc). The controller process mediates interactions with the file system, window data, etc.

Within the scope of the application, we have exposed a global variable (`window.api`) which allows them to communicate in a safe way with important tools that the Candoia platform provides via the controller process. An example of such communication appears below where an app is asking the controller process to run a Boa program and show its output in the content window. This would be a typical 'getting started' step for a Candoia app, because a researcher would first focus on their logic.

```
<h2> My First Candoia Application </h2>
  <div id='content'></div>
  <script>
    var data = api.boa.run('myprog.boa');
    document.getElementbyId('content').
        innerHTML(JSON.stringify(data, null, '\t'));
  </script>
```

This assumes that app also has a file called 'myprog.boa', which is also a valid Boa program.

Libraries available to a Candoia app.

A Candoia app can access several libraries that are exposed to it through the `window.api` variable (in a safe way). These include:

- Running MSR queries (`api.boa`)
- Reading (not writing) files within app (`api.fs`)
- Saving arbitrary data between instances (`api.store`)
- Getting its own package info such as current version (`api.meta`)
- Inter-Process-Communcation handle (`api.ipc`)
- Using pre-made views/graphs. (`api.view`)

The `api.store` is used to save data between multiple runs of the same app. An example appears below.

```
var now = new Date;
api.store.save('last-ran', now);
var data = api.store.get('last-ran');
console.log(data); // "Fri Aug 28 2015 21:23:05 GMT-0500 (CDT)"
```

[8]Currently closed-source apps are not supported.

Writing a Candoia App's MSR Logic.

A Candoia app can access the VCS data, AST data, issue data, team and organization data by way of its MSR logic code. The Javascript glue code is not permitted to directly access this data, but it can access results. Candoia's language for writing an app's MSR logic is an extension of the Boa language [25]. Boa is a domain-specific language specifically introduced for MSR, and its previously proposed implementation runs within an infrastructure hosted at the URL `http://boa.cs.iastate.edu`. As we discuss later, we build on the Boa language specification and have enhanced it further. We have also created an entirely new implementation of the DSL to run on a single node as opposed to a Hadoop cluster, which was the primary execution target of the previous work [25].

Boa language provides domain-specific types for representing project's metadata, revision history, as well as source code. For completeness, we present an example from the Boa website below.

```
1 # what are the churn rates for all projects
2 p: Project = input;
3 counts: output mean[string] of int;

5 visit(p, visitor {
6   before node: Revision ->
7     counts[p.id] << len(node.files);
8 });
```

Line 1 in this program is a comment, line 2 declares that this program accepts a project as input, line 3 declares that this program produces a map of int as output using the mean aggregator. Boa programs can use domain-specific types like `Project`. On line 5-7 is the logic of this program that specifies a traversal of the data organized as a tree rooted at `p`. The code on line 6-7 runs when this traversal encounters a data element of type `Revision`, and if so the code on line 7 says to output the number of files revised in that revision to the aggregator `counts` along with the `id` of the project. The `counts` aggregator aggregates all received values, computes the mean of those values, which is the output of this program. As can be seen from this code, the DSL abstracts away details such as reading project information, reading revision information, data traversal code, data aggregation code, etc, and thus these programs are much shorter, and potentially easier to customize (although customizability of Boa programs has not been demonstrated thus far).

We have extended the set of domain-specific types provided by Boa to also include new types for representing issues. The type for issues provides several kinds of information e.g. the title of the issue, the kind of issue, its priority, its severity, whether it is assigned, the person that has opened, closed, created, or updated this issues, and a list of comments made about the issue.

Candoia Evaluation Engine.

The previous work on the Boa language has created an implementation of the DSL that runs on a Hadoop cluster to be able to process thousands of open source projects from a fixed dataset. For this work on Candoia, we needed another implementation that (1) could run on a single node, (2) be able to read and process local and remote projects, and (3) provide the Candoia platform fine-grained control over its execution, e.g. to start and stop. To satisfy these three goals, we have created an interpreter-based realization of Boa, which runs on a single node. This realization supports and provides all of the logic that is typically abstracted away in Boa programs such as reading project information, reading revision information, data traversal code, data aggregation code, domain-specific functions, etc. This realization also supports our security goals because the platform maintains control over the program counter at all times, and user-defined functions never have control.

This realization was tested using test cases for the original Boa implementation and their expected results. An additional advantage of this realization is that it could be used to run Boa programs outside the Boa infrastructure, which was not possible previously.

Candoia Data Integration Components.

The Candoia platform provides integration with several data sources ranging from local and remotely hosted version control systems, code written in various languages, four different bug repositories (Bugzilla, Jira, Git Issues, and SourceForge tickets), and team and user data for a project. Like Boa we use Protocol Buffers to store data, but compared to Boa in Candoia platform we provide integration with several more data sources. Protocol buffers are a data description format developed by Google that are stored as binary messages. This format is designed to be compact and fast to parse, compared to other formats such as XML. Messages are defined using a struct-like syntax and a compiler is provided which generates Java classes to read and write messages in that format.

Candoia Exchange.

Candoia exchange, a web platform for sharing Candoia apps, is an important aspect of this work. As mentioned previously, our current prototype is a web-based categorized listing of apps that provides information about their Git URL as well as meta-information about the app itself. A Candoia platform can connect to this exchange to gather information about available apps.

4. EVALUATION

Our first study is about applicability of Candoia infrastructure, where we show how researchers and practitioners can write Candoia apps for mining their project management artifacts. Our second study is about adoptability of research using Candoia infrastructure, where we show that researchers and practitioners can easily adopt research prototypes proposed by other researchers (expressed as Candoia apps) without much efforts. Our third study is about customizability and interoperability of expressing research prototypes using Candoia. This study shows that customizations such as enhancing the research prototype, applying the research prototype to different datasets, etc, can be performed without much efforts. Finally, we perform a study of scalability that shows Candoia platform's applicability on large projects.

4.1 Applicability

Our claim is that interesting mining research tasks and hypothesis can be expressed and evaluated using Candoia platform's capabilities. To evaluate the applicability of Candoia, we have created apps for a set of mining research tasks and hypothesis that were of interest to software engineering practitioners (includes hypothesis that were previously proposed by researchers).

Table 1 describes our list of mining research tasks and hypothesis categorized into four categories: I) Bugs, II) Software Evolution, III) Project Management, and IV) Source code analysis and Programming practices. The Candoia apps for these tasks are executed on a set of widely known open-source projects listed in Figure 3 (hereon called test projects). The configuration of the machine used in our experiments consists of an 8-core system (1.6GHz Intel Core i5 Processor) with 8GB 1600MHz DDR3 RAM, 1536MB Intel HD 6000 Graphics card running on OS X Yosemite 10.10.2 and Java 1.8.0_45 with default max heap size. Table 1 shows the execution times of running various Candoia apps on test projects. We haven't spent any time on optimizing these apps for performance yet, so further efficiency gains can be expected in future. More de-

Table 1: Several Candoia apps with their execution times (in seconds).

#	Candoia App	Execution time (s)									
	I. Bugs	Tomcat	Hadoop	JUnit	SLF4j	Bootstrap	Nodejs	Grunt	JQuery	PMD	JEdit
1	Will-never-be-fixed Bugs Detector (Reports on unreproducible or wontfix bugs in project)	30.68	110.53	5.99	2.69	40.51	149.02	2.11	10.13	20.67	47.55
2	Null Detector (Checks the improper use of null)	33.07	152.42	5.86	3.59	4.87	26.3	1.16	3.30	35.86	89.43
3	Check Double-checked (Checks and warns against the use of double cheked locking idiom)	17.04	74.03	3.37	1.64	4.23	24.4	3.04	1.19	15.02	55.46
4	Wait-Notify Police (Checks and warns against improper usage of wait-notify features)	8.17	28.44	2.37	1.23	2.58	12.28	1.87	0.94	8.98	23.14
5	Null Checks Revisions (Identifies bug fixing revisions that added null checks)	3.59	8.15	1.46	2.10	4.72	23.49	5.02	1.47	3.83	5.21
	II. Software Evolution										
6	Hot Files (Lists files that went through most revisions)	28.73	114.29	5.95	26.23	35.7	124.7	2.26	10.95	19.11	57.24
7	Impactful Commits (Lists commits that involved a large number of files)	36.10	124.19	7.84	4.03	43.9	108.31	2.92	12.54	23.29	48.93
8	Commit Blame Assignment (Commit blame assignment based on increase in repository size)	60.84	163.36	9.82	4.74	61.96	189.39	3.18	19.64	32.42	89.58
9	Revision Watcher (Provides details about latest revision, which includes number of files modified, revision date etc.)	32.97	95.11	7.01	3.15	36.90	100.88	2.61	12.20	20.25	48.16
10	Developer Activity Checker (Provides details about developer's last commit)	42.71	139.87	11.8	9.13	48.15	119.24	8.25	17.68	28.41	92.68
	III. Project Management										
11	Top Performers (List top 5 developers who have made most revisions)	31.7	111.64	5.44	2.63	42.19	137.44	2.54	11.38	22.01	46.42
12	Module-developer Mapper (Provides the mapping of modules to developers. Also provides developer and revision information.)	37.29	127.62	7.16	3.94	46.49	171.11	2.56	12.09	24.82	53.00
13	NOA Calculator (Computes number of attributes (NOA))	5.01	19.39	1.77	1.08	2.34	9.37	0.69	1.38	5.45	10.29
14	NPM Calculator (Computes number of public methods (NPM))	1.05	23.84	2.07	6.52	2.23	9.15	0.75	1.59	6.07	6.15
15	Commit Log Police (Checks and identifies developers writing empty or one word commit logs, e.g. fixed)	31.30	110.71	6.41	2.57	35.79	128.77	2.41	10.92	34.90	46.82
	IV. Source code analysis & Programming practices										
16	Convention Checker (Checks naming conventions, e.g. class names should start with an uppercase letter)	10.74	37.92	0.73	1.85	2.50	18.42	1.25	0.39	15.28	22.79
17	Serialization Police (Checks serialization-related properties, e.g. serialVersionUID)	7.35	23.31	3.51	1.54	2.58	9.56	0.79	1.65	33.00	16.98
18	Constant Enforcer (Checks proper declaration of constants, e.g. static field public but not final)	7.42	28.68	2.86	1.28	2.55	9.91	0.71	1.49	9.43	15.74
19	Deadcode Detector (Identifies locations of deadcode)	18.22	110.78	4.77	2.20	4.33	31.66	1.11	4.42	21.62	77.06
20	Flawed Logic Detector (Checks and warn against deeply nested if statements)	11.95	43.57	2.87	1.41	2.65	13.91	0.93	2.07	11.48	33.89
21	Popularity Metrics (Computes various metrics, such as CK Metrics (NOC WMC), OO Metrics (NOPM, FanOut, LOC) etc.	30.45	68.54	3.78	2.07	2.37	14.88	0.87	1.83	31.31	44.37

tailed descriptions of these apps along with their source code is made available via the Candoia website[2] for interested readers.

4.1.1 Interesting Results

Our applicability claim is that interesting mining research tasks can be expressed and evaluated using the Candoia platform. We evaluate this claim by running the Candoia apps listed in Table 1 on test projects and discuss some of the interesting results that our apps produced as a result. Note that, analyzing the results to draw conclusions is not our objective.

Null Checks Revisions. This app identifies bug fixing revisions that added `null` checks. We found a large number of such revisions in test projects. Figure 4 shows the relative number of null checking revisions. For some projects, the frequency of these fixes is quite significant, and for others e.g. Hadoop its quite surprising to see very low number of fixes.

Top Performers. Researchers have studied the developer contribution and other organizational metrics to analyze their influence on software quality. In the same vain, this app lists developers based on their contributions in terms of commits. Figure 5 shows the de-

Projects	Size (KB)	#Files	PL #Files	#LOC	PL #LOC	#AST	#Revs	#Bugs	#Devs
Tomcat 8.0.24	4900	2633	2071	366459	285891	1209368	15493	3023	29
Hadoop 2.7.1	3300	6807	6287	1758594	947891	7490096	11529	10333	49
JUnit 4	68	421	395	29809	26456	164754	2066	148	121
SLF4j 1.7.12	18200	1051	222	151676	137650	67867	1330	332	53
Bootstrap 3.3.5	2500	281	54	61513	28447	197348	11519	213	700
Node.js 0.12.7	542	6006	2209	1533926	201215	1759485	10762	955	30
Grunt 0.4.6	680	39	33	3509	3458	22172	1312	155	20
JQuery 2.1.4	20100	228	155	40626	38429	160602	5882	165	86
PMD 5.3.3	2700	1757	1232	150435	73312	1330824	7947	1394	77
JEdit 5.2.0	3100	1004	573	214126	118492	580194	24025	3926	7

Figure 3: Evaluation projects with details.

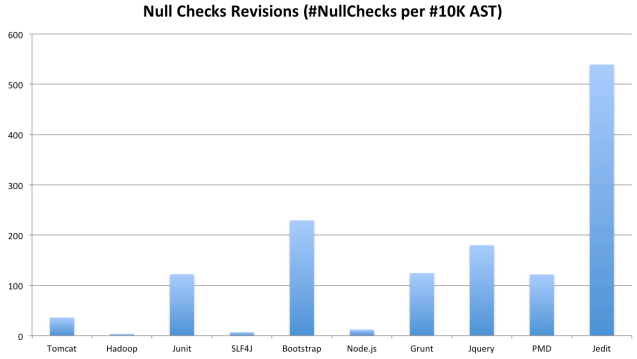

Figure 4: Null check revisions found in open-source projects using Candoia app.

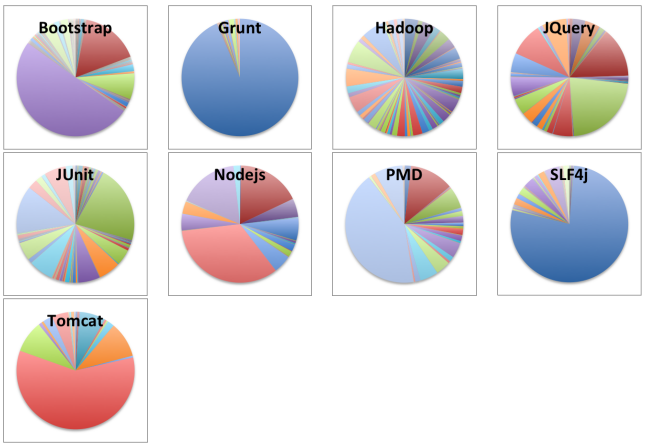

Figure 5: Developer contributions.

veloper contributions for each project in our dataset. Observe that in *Grunt* project most of the commits are done by a single developer, in *Bootstrap*, *SLF4j* and *Tomcat* more than 50% of the overall commit is done by a single developer. *Hadoop* project has equal contributions from many developers.

Table 2: Organizational metrics

NOE	Number of developers who contributed to the component
EF	Component edit frequency
DMO	Group of developers with 70% or more edits to component

Nagappan *et al.* [61] proposed a set of organizational metrics to analyze the influence of organizational structure on software quality. We have created a Candoia app that computes a subset of these metrics shown in Table 2. Nagappan *et al.* have shown that software quality can be analyzed using the values of these metrics. For instance, the metric *NOE* that counts the number of developers who contributed to the component is used to reason about the software quality as follows. The more people who touch the code the lower is the quality. In other words, higher the *NOE* the lower is the quality (more bugs). Similarly other metrics have influences on the software quality. Our Candoia app, which implements this technique outputs the values of the organizational metrics for the project. Using the values of these metrics, one can related it to the bugs in their project. Figure 6 shows the values for *NOE*, *EF* and *DMO* metrics along with the number of bugs in the projects. For quite a few projects there is a strong correlation between bugs and the EF metrics.

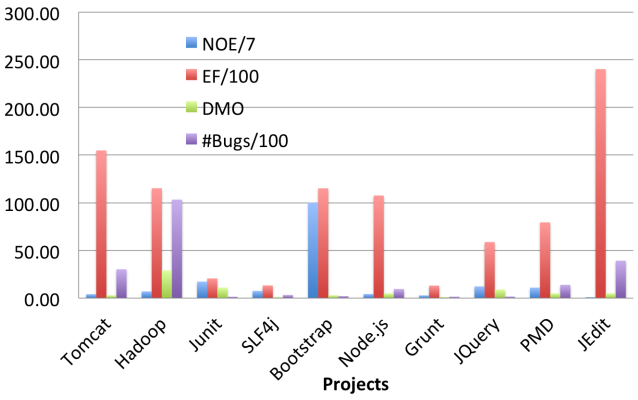

Figure 6: Influence of organizational metrics *NOE*, *EF* and *DMO* on software quality.

Relative Code Churn Metrics. Nagappan and Ball [59] hypothesized that increase in relative code churn measures is accompanied by an increase in system defect density. Inspired from this work, we have built *Relative Code Churn Metrics* app that computes relative code churn measures and outputs the variation in these measures over the revisions. Figure 7 shows this output for our test project *SLF4j*. Figure 7 shows that all measures were steady during project revisions. Using the hypothesis of Nagappan and Ball, we can say that the defect density of *SLF4j* was consistent throughout its development. The development of this app is still in progress, and we have not been able to test it for all of our projects,

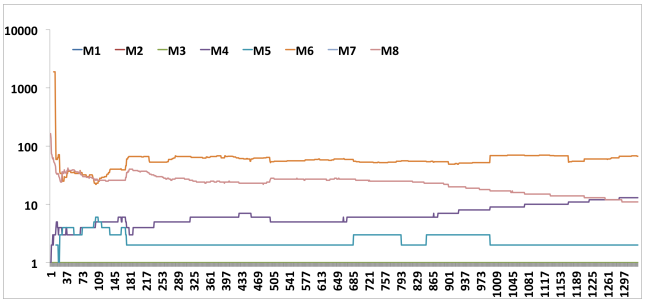

Figure 7: Plot of relative code churn metrics over project revisions using our *Relative code churn metrics* app.

which is the main reason for not including it in Table 1.

Popularity Metrics. Researchers have found that measures of a class often indicates its popularity [19, 5]. These measures include method count, number of attributes, couplings etc. Researchers have proposed a number of metrics for indicating the popularity of classes. For instance, Chidamber and Kemerer [19] proposed CK Metrics. Inspired from these metrics, we have built *Popularity Metrics* app that computes CK Metrics (NOC, WMC), OO Metrics (NOPM, FanOut) etc. The output of *Popularity Metrics* indicates popularity of classes. For instance, high NOC (NOC is number of immediate sub-classes of a class) indicates high reusability, requires more testing and fewer defects due to high reuse. A high WMC (weighted methods per class) has been found to lead to more faults. These were evident in the output of *Popularity Metrics* app for our test projects. For instance, output for *PMD* project, indicated that JavaParser class had higher WMC compared to all other classes (58, recommended WMC is at max 24). This indicates that the JavaParser class is expected to have more detects, which was indeed the case when we cross checked *Hot Files* app results, where JavaParser was in the top five revised files.

A number of Candoia apps are written to detect problems in programming practices (*Convention Checker*, *Serialization Police*, *Constant Enforcer*), concurrency (*Check Double-checked*, *Wait-notify Police*), logic (*Flawed Logic Detector*), optimization (*Deadcode Detector*), bad assumptions (*Null Detector*) etc. Using these apps we found that, our test projects strictly abide to the coding conventions. For instance, *Convention Checker*, for identifying the problems in coding convention returned no result for these test projects (but worked for intentionally introduced convention violations). This is expected, as well maintained projects use good programming practices. Candoia apps that checks for bug patterns didn't find double-checked locking or bad wait-notify idioms, but found instances of non-null assumptions where the methods do not check for null arguments (in *Tomcat*, *Hadoop*, *JUnit* projects). Candoia App *Flawed Logic Detector* found 300 instances of using `if-else -if` nesting with nesting levels more than 10 in Apache Tomcat.

In summary, various interesting research prototypes can be expressed as Candoia apps, evaluated using the Candoia platform and shared with researchers and practitioners.

4.2 Adoptability

The goal of this study is to demonstrate that SE practitioners can adopt a research prototype as-is to their own project environment using the Candoia platform.

In this study we conduct a set of experiments consisting of running the Candoia apps of research prototypes listed in Table 1 on different project environments (or project configurations). Project environment defines the version control system, programming language, bug tracking and information about other supporting tools. Table 3 lists different project environments in our test projects.

Table 3: Different project environments used in our adoptability experiments.

Projects	VCS	PL	Bug Tracking
Tomcat 8.0.24	SVN	Java	Bugzilla
Hadoop 2.7.1	GIT	Java	Jira
JUnit 4	GIT	Java	Git Issues
SLF4j 1.7.12	GIT	Java	Jira
Bootstrap 3.3.5	GIT	JS	Git Issues
Node.js 0.12.7	GIT	JS	Git Issues
Grunt 0.4.6	GIT	JS	Git Issues
JQuery 2.1.4	GIT	JS	Git Issues
PMD 5.3.3	GIT	Java	SF Tickets
JEdit 5.2.0	SVN	Java	SF Tickets

Our claim is that, being able to run Candoia apps in different project environments without requiring to change the research prototype (Candoia app) shows that research prototypes can be easily adopted. Table 1 shows the execution times of various Candoia apps. Running the Candoia apps listed in Table 1 on different project environments did not require any changes to the app. This demonstrates that research prototypes (expressed as Candoia apps) can be easily adopted.

4.3 Customizability and Interoperability

Our claim is that small customizations of a Candoia app to suit project's need better are easy. For demonstrating, we performed a user study as described below.

User study setting. We gathered a group of eight Candoia app developers with varying expertise. We determine the developer expertise by asking several questions such as, *how many years of experience do you have with GIT/SVN/CVS/Perforce?*, *how much experience do you have configuring, building and installing tools?* etc. We then asked the developers to customize a Candoia app provided to them. Each developer performs the tasks described in Table 4 (in order).

Table 4: Customization task steps

#1	*Answers the questionnaire about their background*
#2	*Selects a Candoia app and a project configuration from the list of project configurations*
#3	*Runs the Candoia app on the project configuration*
#4	*Customizes the Candoia app based on the customization requirement provided to them*
#5	*Re-runs the customized Candoia app on the previously selected project configurations*
#6	*Also runs the customized Candoia app on a new project configuration*
#7	*Answers the questionnaire at the end of the task.*

In our customization user study, developers are asked to select a Candoia app of their interest from the following three Candoia apps: i) *Will-never-be-fixed Bug Detector*, ii) *Hot Files* and iii) *NOA Calculator*.

Task #1. Customizing *Will-never-be-fixed Bug Detector*. This app reports on unreproducible or wontfix bugs in the project. The customization task is to include the duplicate bug reports as well in the result.

Task #2. Customizing *Hot Files*. This app lists files that went through most revisions. The customization task is to apply year filter to list hot files in year 2010.

Task #3. Customizing *NOA calculator*. This Candoia app computes a software metric NOA (number of attributes per type). The customization task is to compute NOA per revision. This customization task will help one to analyze how NOA changed over project revisions.

The following three project configurations were used in our experiment:

#	Project	VCS	PL	Bug Tracking
1	Bootstrap 3.3.5	GIT	JS	Git Issues
2	JUnit 4	GIT	Java	Git Issues
3	Tomcat 8.0.24	SVN	Java	BugZilla

These project environments helps us ensure that customized Candoia apps are also adoptable. As part of our customization task, step #6 asks the developers to run their customized Candoia app on a new project configuration that was not used before.

Observations. We recorded developer responses to background questionnaire and Candoia experience questionnaire. We also recorded the time they took to complete the customization task. Figure 7 shows the recorded responses.

B1	*Industry experience?*
B2	*GIT/SVN/CVS/Perforce experience?*
B3	*BugZilla/Git Issues experience?*
B4	*Configure, build and install tools experience*
	0-1years, 1-2 years, 2-4 years, more than 4 years

Table 5: Developer background questionnaire.

E1	*How easy or difficult it is to run a Candoia app on your project?*
E2	*How easy or difficult it is to customize?*
E3	*How easy or difficult it is to run your customized Candoia app on a different project?*
	0-Very Easy, 1-Easy, 2-Moderate, 3-Difficult, 4-Complex

Table 6: Candoia customization experience questionnaire.

Table 7: Responses recorded from eight developers. The background questions B1-B4 is described in Table 5 and the Candoia experience related questions are described in Table 6.

Developer	Background				Candoia Experience			Task time
#	*B1*	*B2*	*B3*	*B4*	*E1*	*E2*	*E3*	(min)
1	4	3	1	1	0	1	1	12
2	0	4	1	4	1	2	1	30
3	2	1	1	1	0	2	0	44
4	2	2	1	1	1	1	0	16
5	1	3	1	2	2	1	1	15
6	1	1	1	1	1	0	1	13
7	1	1	1	1	1	1	1	15
8	1	2	2	4	0	2	0	40

From Table 7 it can be seen that developers with different levels of experiences in terms of industry experience, GIT/SVN/CVS tools experience and support tool experience, are considered. Except one developer all others found it easy to run the Candoia app on their selected project (E1) and run the customized Candoia app on a new project (E3). However, three of the eight developers found it difficult to perform the customization task (developers #2, #3 and #8), which is reflected in the Candoia experience question E2 and the time they took to complete the task. These developers mentioned the hurdles they had in the comments section of their responses. Lack of MSR expertise and lack of debugging facilities were the two main hurdles for these developers. Apart from these three developers, others could finish the customization task in about 15 minutes. In these 15 minutes, developers were able to run the Candoia app of their selection on their project, customize the app and re-run the app on a new project (that has different configuration than the original). In summary we believe that this study is a good smoke test of Candoia's usability, customizability and adoptability.

4.4 Scalability

We evaluate scalability of Candoia using the dataset previously described in Figure 3. This dataset consists of ten popular projects with project data sizes varying between 3MB to 600MB.

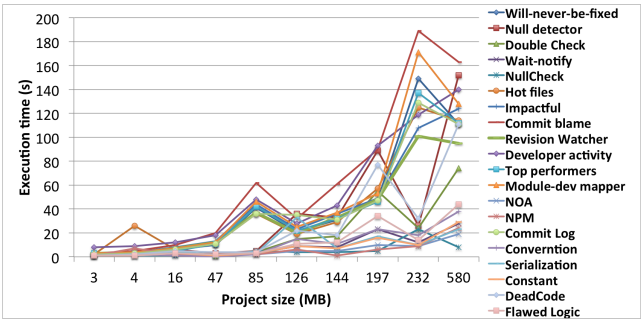

Figure 8: Scalability of Candoia

Figure 8 shows the scalability of Candoia, where we plot the execution times of various Candoia apps for our test projects. The X-axis shows project sizes in log-scale. From this chart we can observe that the running time of these apps does increase with larger project size, however, even for the largest project all apps are able to complete their tasks in 3 minutes or less. Future work on the Candoia platform as well as on the apps will focus on improving scalability as one of our key goals.

5. RELATED IDEAS

Our idea of a platform and an ecosystem for bridging the gap between MSR research and practice is new; however, we draw inspiration from a rich body of work in this area. In terms of its focus, the Candoia platform is closer to Bevan *et al.*'s *Kenyon* [9], Bajracharya *et al.*'s *Sourcerer* [6], Gousios and Spinellis's Alitheia Core [32, 31], Howison *et al.*'s *FLOSSMole* [39] and different from Boetticher *et al.*'s PROMISE Repository [69], González-Barahona and Robles's open-access data repositories [29], Black Duck OpenHub (aka Ohloh) [13], GHTorrent [30, 33], Ossher *et al.*'s SourcererDB [64], the SourceForge Research Data Archive (SRDA) [28], and Boa [25]. The former set of approaches provide frameworks for MSR, whereas the latter set of approaches provide a repository of datasets from open source projects, which eases MSR tasks because researcher's do not have to collect and curate datasets [57]. Both Kenyon and Sourcerer define database schemas for metadata and source code and provide access to this dataset via SQL. Later on

Grechanik *et al.* [35] have also taken this approach. Their approach easily supports joins on the data, whereas APIs for writing Candoia apps do not currently have support for join. Furthermore, Candoia also provides revision information, access to fine-grained program elements, support for multiple languages, issue, and personnel data. Compared to Boa, which also provides revision information and access to fine-grained program elements but on a very large, but fixed dataset from open source repositories, Candoia provides facilities to analyze a developer or organization's private projects. Boa also does not support source data from multiple languages, issue data, and team data as of this writing.

Alitheia Core's goal is to provide a highly extensible framework for analyzing software product and process metrics on a large database of open source projects' source code, bug records and mailing lists. Applications using Alitheia Core are written using Java and use built-in plug-ins and/or create new plug-ins to compute their desired metrics. Alitheia Core also abstracts raw data into object-relational entities and includes source code revisions as in our framework. However, the main purpose of GHTorrent and Alitheia Core is different from Candoia. While Alitheia Core focuses on software metrics and GHTorrent focuses on event streams, Candoia provides fine-grained program elements, i.e. AST nodes and mechanisms for source code traversal. Similarly, FLOSSMole gathers metadata (e.g., project topics, activity, statistics, licenses, developer skills etc), whereas Candoia also provides access to source code and revision history.

Pinto *et al.*'s Groundhog [65] is an infrastructure for downloading projects from SourceForge, analyzing the source code, and collect metrics from these projects. It can facilitate learning the Java open source projects hosted by SourceForge. It does not cover other popular programming languages, such as Javascript, does not study other source code repositories, e.g., Github and Google Code, and does not have support for issues and team data. Also the Groundhog infrastructure cannot be used to study the evolution of the projects.

Also, closely related to Candoia is work on Soot, a program analysis framework [80]; LLVM, a compilation framework [44]; Tricorder, a program analysis ecosystem and its open source implementation SHIPSHAPE [73], and Joern [87], a platform for robust analysis of C/C++ code. All of these framework, except Tricorder and Joern which are recently emerging but showing significant promise, have had success in bridging the gap between research and practice. While Soot, LLVM, Tricorder, and Joern focus on program analysis and compilation, Candoia focuses on mining software repositories, with emphasis on software evolution.

Candoia is related to Android, iOS platform, and Atom[1] platform in terms of its approach, but has different goals.

6. CONCLUSION AND FUTURE WORK

Over the last decade Mining software repositories research has produced a large number of significant results that can help software engineering in practice. There is, however, a valley of death between these inventions and their deployment in practice. In this work, we present Candoia, a platform and an ecosystem to ease transition of MSR research into practice. We have implemented both the Candoia platform and the Candoia ecosystem, which allows researchers to develop their research as apps and publish them. We have already developed over two dozen apps in four different categories. Our evaluation demonstrates that Candoia substantially reduces the cost of preparing MSR ideas for use in practice. The need to provide support for a wide variety of tools and technologies e.g. CVS, SVN, Git, Bugzilla, Jira, Issues, etc, to improve applicability is mitigated. Candoia also makes using same apps for variety of project configurations easier. Furthermore, small customizations of a Candoia app to suit project's need better are easy.

In the future, we plan to integrate additional tools and technologies with the Candoia platform to further improve its applicability. We believe that the main challenge to that end would be define extensible interfaces that can allow us and others to plugin support for additional tools. We are also very excited about new apps that us and others can develop for the platform. Last but not least, we plan to further improve our evaluation engine to decrease app runtime and improve scalability.

Acknowledgment

This work was supported in part by the US National Science Foundation under grants CCF-15-18897, CNS-15-13263, and CCF-14-23370. Ramanathan Ramu helped with the Candoia exchange website, and Trey Erenberger helped with the Candoia frontend code. Robert Dyer helped explain semantics of Boa's generated code, and Hoan Nguyen helped explain VCS libraries. The authors would also like to thank Tien N. Nguyen, Robert Dyer, Hoan A. Nguyen, and Kathryn Stolee for feedback on these ideas.

7. REFERENCES

[1] M. Acharya, T. Xie, J. Pei, and J. Xu. Mining API patterns as partial orders from source code: from usage scenarios to specifications. *ESEC/FSE '07*, pages 25–34. 2007.

[2] R. Alur, P. Černý, P. Madhusudan, and W. Nam. Synthesis of interface specifications for Java classes. *POPL '05*, pages 98–109. 2005.

[3] G. Ammons, R. Bodík, and J. R. Larus. Mining specifications. *POPL '02*, pages 4–16. 2002.

[4] J. Anvik, L. Hiew, and G. C. Murphy. Who should fix this bug? *ICSE '06*, pages 361–370, New York, NY, USA, 2006.

[5] A. Bacchelli, M. D'Ambros, and M. Lanza. Are popular classes more defect prone? *FASE*, pages 59–73, 2010.

[6] S. Bajracharya, J. Ossher, and C. Lopes. Sourcerer: An infrastructure for large-scale collection and analysis of open-source code. *Sci. Comput. Program.*, 79:241–259, Jan. 2014.

[7] V. R. Basili, L. C. Briand, and W. L. Melo. A validation of object-oriented design metrics as quality indicators. *IEEE Trans. Softw. Eng.*, 22(10):751–761, 1996.

[8] O. Baysal, M. Godfrey, and R. Cohen. A bug you like: A framework for automated assignment of bugs. *ICPC '09*, pages 297–298, May 2009.

[9] J. Bevan, E. J. Whitehead, Jr., S. Kim, and M. Godfrey. Facilitating software evolution research with kenyon. *ESEC/FSE-13*, pages 177–186, New York, NY, USA, 2005.

[10] P. Bhattacharya and I. Neamtiu. Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging. *ICSM '10*, pages 1–10, Washington, DC, USA, 2010.

[11] C. Bird, N. Nagappan, P. Devanbu, H. Gall, and B. Murphy. Does distributed development affect software quality? an empirical case study of windows vista. *ICSE '09*, pages 518–528. 2009.

[12] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu. Don't touch my code!: examining the effects of ownership on software quality. *ESEC/FSE '11*, pages 4–14. 2011.

[13] Black Duck Software. Black duck open HUB. `https://www.openhub.net/`, 2015.

[14] L. C. Briand, J. Wust, S. V. Ikonomovski, and H. Lounis. Investigating quality factors in object-oriented designs: an

industrial case study. *ICSE '99*, pages 345–354. 1999.

[15] G. Canfora and L. Cerulo. How software repositories can help in resolving a new change request. *Workshop on Empirical Studies in Reverse Engineering*, 2005.

[16] G. Canfora and L. Cerulo. Supporting change request assignment in open source development. *SAC '06*, pages 1767–1772. 2006.

[17] M. Cataldo, A. Mockus, J. A. Roberts, and J. D. Herbsleb. Software dependencies, work dependencies, and their impact on failures. *IEEE Transactions on Software Engineering*, 99:864–878, 2009.

[18] R.-Y. Chang, A. Podgurski, and J. Yang. Discovering neglected conditions in software by mining dependence graphs. *IEEE Trans. Softw. Eng.*, 34(5):579–596, 2008.

[19] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Trans. Softw. Eng.*, 20(6):476–493, June 1994.

[20] D. Cubranic and G. Murphy. Automatic bug triage using text categorization. *SEKE 2004*, pages 92–97. 2004.

[21] V. Dallmeier, C. Lindig, and A. Zeller. Lightweight Defect Localization for Java. *ECOOP 2005*. 2005.

[22] V. Dallmeier, A. Zeller, and B. Meyer. Generating fixes from object behavior anomalies. *ASE'09*, pages 550–554. November 2009.

[23] M. D'Ambros, M. Lanza, and R. Robbes. Evaluating defect prediction approaches: a benchmark and an extensive comparison. *Empirical Softw. Engg.*, DOI: 10.1007/s10664-011-9173-9, 2011.

[24] G. Di Fatta, S. Leue, and E. Stegantova. Discriminative pattern mining in software fault detection. *SOQUA '06*, pages 62–69. 2006.

[25] R. Dyer, H. A. Nguyen, H. Rajan, and T. N. Nguyen. Boa: A language and infrastructure for analyzing ultra-large-scale software repositories. *ICSE '13*, pages 422–431. 2013.

[26] D. Engler, D. Y. Chen, S. Hallem, A. Chou, and B. Chelf. Bugs as deviant behavior: a general approach to inferring errors in systems code. *SOSP '01*, pages 57–72. 2001.

[27] M. Gabel and Z. Su. Javert: fully automatic mining of general temporal properties from dynamic traces. *SIGSOFT '08/FSE-16*, pages 339–349. 2008.

[28] Y. Gao, M. V. Antwerp, S. Christley, and G. Madey. A research collaboratory for open source software research. *FLOSS '07*, pages 4–, Washington, DC, USA, 2007.

[29] J. M. González-Barahona and G. Robles. On the reproducibility of empirical software engineering studies based on data retrieved from development repositories. *Empirical Software Engineering*, 17(1-2):75–89, 2012.

[30] G. Gousios. The GHTorrent dataset and tool suite. *MSR '13*, pages 233–236. 2013.

[31] G. Gousios and D. Spinellis. Alitheia core: An extensible software quality monitoring platform. *ICSE '09*, pages 579–582. 2009.

[32] G. Gousios and D. Spinellis. A platform for software engineering research. *MSR'09*, pages 31–40, 2009.

[33] G. Gousios and D. Spinellis. GHTorrent: GitHub's data from a firehose. *MSR '12*, pages 12–21. 2012.

[34] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy. Predicting fault incidence using software change history. *IEEE Trans. Softw. Eng.*, 26(7):653–661, 2000.

[35] M. Grechanik, C. McMillan, L. DeFerrari, M. Comi, S. Crespi, D. Poshyvanyk, C. Fu, Q. Xie, and C. Ghezzi. An empirical investigation into a large-scale java open source code repository. *ESEM '10*, page 11. 2010.

[36] A. E. Hassan. Predicting faults using the complexity of code changes. *ICSE '09*, pages 78–88. 2009.

[37] T. A. Henzinger, R. Jhala, and R. Majumdar. Permissive interfaces. *ESEC/FSE-13*, pages 31–40. 2005.

[38] D. Hovemeyer and W. Pugh. Finding bugs is easy. *SIGPLAN Not.*, 39(12):92–106, 2004.

[39] J. Howison, M. Conklin, and K. Crowston. Flossmole: A collaborative repository for floss research data and analyses. *IJITWE '06*, 2006.

[40] G. Jeong, S. Kim, and T. Zimmermann. Improving bug triage with bug tossing graphs. *ESEC/FSE '09*, pages 111–120, New York, NY, USA, 2009.

[41] T. M. Khoshgoftaar and E. B. Allen. Ordering fault-prone software modules. *Software Quality Control*, 11(1):19–37, 2003.

[42] S. Kim, K. Pan, and E. E. J. Whitehead, Jr. Memories of bug fixes. *SIGSOFT '06/FSE-14*, pages 35–45. 2006.

[43] S. Kim, T. Zimmermann, J. Whitehead, and A. Zeller. Predicting faults from cached history. *ICSE '07*, pages 489–498. 2007.

[44] C. Lattner and V. Adve. Llvm: A compilation framework for lifelong program analysis & transformation. *CGO '04*, pages 75–86, 2004.

[45] T. Lee, J. Nam, D. Han, S. Kim, and H. P. In. Micro interaction metrics for defect prediction. *ESEC/FSE '11*, pages 311–321. 2011.

[46] Z. Li, S. Lu, and S. Myagmar. Cp-miner: Finding copy-paste and related bugs in large-scale software code. *IEEE Trans. Softw. Eng.*, 32(3):176–192, 2006.

[47] Z. Li and Y. Zhou. PR-Miner: automatically extracting implicit programming rules and detecting violations in large software code. *ESEC/FSE-13*, pages 306–315. 2005.

[48] Z. Lin, F. Shu, Y. Yang, C. Hu, and Q. Wang. An empirical study on bug assignment automation using chinese bug data. *ESEM '09*, pages 451–455, Washington, DC, USA, 2009.

[49] C. Liu, E. Ye, and D. J. Richardson. Software library usage pattern extraction using a software model checker. *ASE '06*, pages 301–304. 2006.

[50] B. Livshits and T. Zimmermann. Dynamine: finding common error patterns by mining software revision histories. *SIGSOFT Softw. Eng. Notes*, 30(5):296–305, 2005.

[51] D. Lo and S. Maoz. Mining hierarchical scenario-based specifications. *ASE'09*, pages 359–370. November 2009.

[52] G. d. Lucca. An approach to classify software maintenance requests. *ICSM '02*, page 93. 2002.

[53] D. Matter, A. Kuhn, and O. Nierstrasz. Assigning bug reports using a vocabulary-based expertise model of developers. *MSR '09*, pages 131–140, Washington, DC, USA, 2009.

[54] A. Meneely, L. Williams, W. Snipes, and J. Osborne. Predicting failures with developer networks and social network analysis. *SIGSOFT '08/FSE-16*, pages 13–23. 2008.

[55] T. Menzies, J. Greenwald, and A. Frank. Data mining static code attributes to learn defect predictors. *IEEE Trans. Softw. Eng.*, 33(1):2–13, 2007.

[56] A. Michail. Data mining library reuse patterns using generalized association rules. *ICSE '00*, pages 167–176. 2000.

[57] A. Mockus. Amassing and indexing a large sample of version control systems: Towards the census of public source

code history. *MSR '09*, pages 11–20, Washington, DC, USA, 2009.

[58] A. Mockus. Organizational volatility and developer productivity. *STC 2009*. 2009.

[59] N. Nagappan and T. Ball. Use of relative code churn measures to predict system defect density. *ICSE '05*, pages 284–292. 2005.

[60] N. Nagappan and T. Ball. Using software dependencies and churn metrics to predict field failures: An empirical case study. *ESEM '07*, pages 364–373. 2007.

[61] N. Nagappan, B. Murphy, and V. Basili. The influence of organizational structure on software quality: an empirical case study. *ICSE '08*, pages 521–530. 2008.

[62] N. Nagappan, A. Zeller, T. Zimmermann, K. Herzig, and B. Murphy. Change bursts as defect predictors. *ISSRE*, pages 309–318. 2010.

[63] T. T. Nguyen, H. A. Nguyen, N. H. Pham, J. M. Al-Kofahi, and T. N. Nguyen. Graph-based Mining of Multiple Object Usage Patterns. *ESEC/FSE 2009*. 2009.

[64] J. Ossher, S. Bajracharya, E. Linstead, P. Baldi, and C. Lopes. SourcererDB: An Aggregated Repository of Statically Analyzed and Cross-linked Open Source Java Projects. *MSR '09*, pages 183–186, Washington, DC, USA, 2009.

[65] G. Pinto, W. Torres, B. Fernandes, F. Castor, and R. S. Barros. A Large-Scale Study on the Usage of Java's Concurrent Programming Constructs. *Journal of Systems and Software*, 106:59–81, 2015.

[66] M. Pinzger, N. Nagappan, and B. Murphy. Can developer-module networks predict failures? *SIGSOFT '08/FSE-16*, pages 2–12. 2008.

[67] A. Podgurski, D. Leon, P. Francis, W. Masri, M. Minch, J. Sun, and B. Wang. Automated support for classifying software failure reports. *ICSE '03*, pages 465–475, Washington, DC, USA, 2003.

[68] M. Pradel and T. R. Gross. Automatic generation of object usage specifications from large method traces. *ASE'09*, pages 371–382. November 2009.

[69] Promise 2009. http://promisedata.org/2009/datasets.html.

[70] M. M. Rahman, G. Ruhe, and T. Zimmermann. Optimized assignment of developers for fixing bugs an initial evaluation for eclipse projects. *ESEM '09*, pages 439–442, Washington, DC, USA, 2009.

[71] M. K. Ramanathan, A. Grama, and S. Jagannathan. Path-sensitive inference of function precedence protocols. *ICSE '07*, pages 240–250. 2007.

[72] J. Ratzinger, M. Pinzger, and H. Gall. Eq-mine: Predicting short-term defects for software evolution. *FASE '07*, pages 12–26. 2007.

[73] C. Sadowski, J. van Gogh, C. Jaspan, E. Soederberg, and C. Winter. Tricorder: Building a program analysis ecosystem. *ICSE '15*, 2015.

[74] A. Schroter, T. Zimmermann, and A. Zeller. Predicting component failures at design time. *ESEM '06*, pages 18–27. 2006.

[75] S. Shoham, E. Yahav, S. Fink, and M. Pistoia. Static specification mining using automata-based abstractions. *ISSTA '07*, pages 174–184. 2007.

[76] R. Subramanyam and M. S. Krishnan. Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects. *IEEE Trans. Softw. Eng.*, 29:297–310, April 2003.

[77] The Chromium Project. Chromium: Open source web browser. www.chromium.org, 2008.

[78] S. Thummalapenta and T. Xie. Parseweb: a programmer assistant for reusing open source code on the Web. *ASE '07*, pages 204–213. 2007.

[79] S. Thummalapenta and T. Xie. Alattin: Mining alternative patterns for detecting neglected conditions. *ASE'09*, pages 283–294. November 2009.

[80] R. Vallee-Rai, P. Co, E. Gagnon, L. Hendren, P. Lam, and V. Sundaresan. Soot-a Java bytecode optimization framework. *CASCON '99*, page 13, 1999.

[81] A. Wasylkowski and A. Zeller. Mining temporal specifications from object usage. *ASE'09*, pages 295–306. November 2009.

[82] A. Wasylkowski, A. Zeller, and C. Lindig. Detecting object usage anomalies. *ESEC-FSE '07*, pages 35–44. 2007.

[83] W. Weimer and G. C. Necula. Mining temporal specifications for error detection. *TACAS '05*, pages 461–476, 2005.

[84] E. J. Weyuker, T. J. Ostrand, and R. M. Bell. Using developer information as a factor for fault prediction. *PROMISE '07*, page 8. 2007.

[85] C. C. Williams and J. K. Hollingsworth. Automatic mining of source code repositories to improve bug finding techniques. *IEEE Trans. Softw. Eng.*, 31(6):466–480, 2005.

[86] T. Wolf, A. Schroter, D. Damian, and T. Nguyen. Predicting build failures using social network analysis on developer communication. *ICSE '09*, pages 1–11. 2009.

[87] F. Yamaguchi, N. Golde, D. Arp, and K. Rieck. Modeling and discovering vulnerabilities with code property graphs. *SP '14*, pages 590–604. 2014.

[88] J. Yang, D. Evans, D. Bhardwaj, T. Bhat, and M. Das. Perracotta: mining temporal API rules from imperfect traces. *ICSE '06*, pages 282–291. 2006.

[89] H. Zhong, L. Zhang, T. Xie, and H. Mei. Inferring Resource Specifications from Natural Language API Documentation. *ASE'09*, pages 307–318. November 2009.

[90] T. Zimmermann and N. Nagappan. Predicting defects using network analysis on dependency graphs. *ICSE '08*, pages 531–540. 2008.