2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)

Candoia: A Platform for Building and Sharing Mining Software Repositories Tools as Apps

Nitin M Tiwari
Iowa State University
nmtiwari@iastate.edu

Ganesha Upadhyaya
Iowa State University
ganeshau@iastate.edu

Hoan Anh Nguyen
Iowa State University
hoan@iastate.edu

Hridesh Rajan
Iowa State University
hridesh@iastate.edu

***Abstract*—We propose Candoia, a novel platform and ecosystem for building and sharing Mining Software Repositories (MSR) tools. Using Candoia, MSR tools are built as apps, and Candoia ecosystem, acting as an appstore, allows effective sharing. Candoia platform provides, data extraction tools for curating custom datasets for user projects, and data abstractions for enabling uniform access to MSR artifacts from disparate sources, which makes apps portable and adoptable across diverse software project settings of MSR researchers and practitioners. The structured design of a Candoia app and the languages selected for building various components of a Candoia app promotes easy customization. To evaluate Candoia we have built over two dozen MSR apps for analyzing bugs, software evolution, project management aspects, and source code and programming practices showing the applicability of the platform for building a variety of MSR apps. For testing portability of apps across diverse project settings, we tested the apps using ten popular project repositories, such as Apache Tomcat, JUnit, Node.js, etc, and found that apps required no changes to be portable. We performed a user study to test customizability and we found that five of eight Candoia users found it very easy to customize an existing app. Candoia is available for download.**

I. Introduction

Over the last decade, mining software repositories (MSR) research has helped make significant advances in software engineering (SE) — defect prediction [4], [12], [30], source code analysis and pattern discovery [26]–[28], [43], [45], mining software specifications [2], [11], [46], [48], social network analysis of software development [7], [10], [29], [32], [36], [47] to name a few. Researchers have shown that further advances can be made if the process of building and widely distributing MSR tools is eased [3], [5], [13], [19], [21], [23]. Toward this end, we propose Candoia, a platform and ecosystem for building and sharing MSR tools. Using Candoia, MSR tools are built as apps, and Candoia ecosystem, acting as an appstore, allows effective sharing of MSR apps. Candoia platform provides data extraction tools and data abstractions for mining MSR artifacts[1]. Candoia provides suitable abstractions for building MSR tools, popularly known as apps.

Candoia's main contribution is the process of building and sharing MSR tools as apps which are portable, adoptable, and customizable for MSR researchers and practitioners. There have been similar efforts along two directions to help MSR researchers and practitioners. First set of approaches provide i) platforms for reusing of tools and allow low cost addition of new tools [13], ii) frameworks that define database schemas for storing MSR artifacts (such as revision history, source code, etc.) and provide access via SQL [3], [5], [19], [21], [23] and iii) infrastructures for downloading projects from open-source repositories, analyzing the source code, revision histories and other MSR artifacts, and building the dataset for testing the hypothesis [24], [35], [35]. The second set of approaches provides a repository of datasets from open-source repositories so that researchers do not have to collect and curate datasets [14], [22], [37]. When compared to the first set of approaches that are mainly focused on enabling faster MSR prototyping, Candoia enables easier building and customizing of MSR tools, and achieves portability of the tools across diverse project settings. When compared to the second set of approaches that are focused on providing standard datasets, Candoia allows mining user specific datasets.

Candoia makes several contributions to ease the process of building and sharing MSR tools by promoting adoptability and customizability. Building MSR tools require building or using pre-built data extraction tools to gather MSR artifacts. Candoia platform provides a large set of data extraction tools for extracting the MSR artifacts from user projects and curating the user datasets. This eases the process of building MSR tools. We have created a robust implementation of the Candoia platform. To evaluate, we have built over two dozen different MSR apps for analyzing bugs, software evolution, project management aspects, and source code and programming practices. A survey of MSR tools found that generalization of MSR tools beyond their subject dataset could make them more replicable and adoptable [38]. In this regard, the Candoia platform provides data abstractions for mining MSR artifacts and these abstractions provide uniform access to MSR artifacts from disparate sources. Since apps are built on top of Candoia's data abstractions and not on top of raw MSR artifacts, apps become portable across diverse project settings. A project setting defines types and sources of various MSR artifacts, such as GIT or SVN version control systems (VCS), Bugzilla, GitHub-Issues, JIRA or SF-Tickets bug tracking, Java or Javascript source files, GitHub or SF.net forges.

For evaluating the portability of apps across diverse project

[1]MSR artifacts include version control system (VCS) data from GIT, SVN, CVS, etc, source code written using programming language(s) such as Java, Javascript, etc, bug data from repositories such as Bugzilla, JIRA, GitHub-Issues, SF-Tickets, etc, project metadata, and users and teams data from forges such as SF.net, GitHub

978-1-5386-1544-7/17 $31.00 © 2017 IEEE
DOI 10.1109/MSR.2017.56

settings, we tested the apps using ten popular projects repositories, which include ApacheTomcat, JUnit, Node.js, etc. These project repositories provided us a variety of project settings to test portability of apps and we found that all of our apps required no change to be able to run on diverse project settings. Researchers and practitioners while adopting an MSR tool, wants to perform few customizations to suit their needs. Candoia promotes easy customizations because of the structured design of Candoia apps and the languages selected for building various components of an Candoia app. We performed a user study consisting of 8 MSR app developers with varying expertise for testing the customizability aspect of Candoia. We found that 5 of 8 developers found it easy to customize an existing Candoia app.

Candoia platform, as well as all of its current two dozen apps, are open-source projects and they are available for download. Sharing a new Candoia app is as simple as creating a new GitHub project and adding app files to that project, and even first year undergraduates have built some apps.

II. Motivation

In this section, we motivate the need for a platform and ecosystem that promotes a process of building MSR tools as light-weight apps that are easily portable and customizable.

Today MSR tools are built for a specific software project setting or a specific dataset. A software project setting describes: 1) the repository (or the forge) where the project is maintained, 2) the programming language(s) used in the project source code, 3) the bug repository, and 4) the version control system (VCS) used for maintaining project revisions. An example project setting of a user that contains JUnit project consists of: GitHub as forge, Java source files, GitHub−Issues for bug tracking, and GIT version control data[2].

Consider a researcher who wants to build an MSR tool *Association Mining* for predicting bugs by mining file associations. If the researcher building this tool uses the JUnit project setting for evaluation, it requires them to build a tool chain (or use existing tools) consisting of: i) GitHub project reader, ii) GIT version data reader, iii) Java parser, iv) GitHub−Issues adapter, for extracting different MSR artifacts to be used in the *Association Mining* tool. The association mining logic uses *Eclat* association algorithm for which the researcher imports Weka library. Overall, the *Association Mining* tool contains the mining logic (the association mining algorithm) that is tightly integrated with the supporting tools for reading and processing the project specific artifacts as shown in Figure 1.

Now consider a practitioner who wants to adopt the *Association Mining* MSR tool and perform few customizations to suite their needs. If the practitioner's project setting is similar to that of the researcher, then the *Association Mining* MSR tool is readily adoptable, otherwise, the practitioner cannot adopt the *Association Mining* MSR tool as is. For instance, if the practitioner's project setting consists of JEdit project with SF .net as forge, Java source files, SF−Tickets as bug repository,

[2]A project setting of an MSR user may include multiple projects, we consider one project for simplifying the illustration.

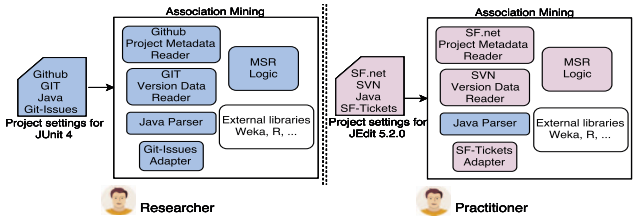

Fig. 1. A scenario of a practitioner adopting a MSR tool built by a researcher.

and SVN version control data. In this scenario, the practitioner has two choices: 1) throw away the *Association Mining* MSR tool, or 2) try to adopt the tool by disintegrating it and making several modifications to it based on their project setting. The practitioner might face one or more of the following challenges when adopting this MSR tool:

1) Reproducibility: The practitioner needs to have access to the tool, its supporting tools and libraries, and the details about how the dataset was curated (often these details are missing [38]). Upon having access to the tool, dataset, and the supporting tools, the practitioner can deintegrate the tool and try to adopt it based on his project settings.

2) Adoptability: The practitioner may not be able to use the tool chain of the researcher because the project settings have changed and they need to build a tool chain (or use existing tools) consisting of: i) SF.net project reader, ii) SVN version data reader, iii) Java parser, iv) SF−Tickets adapter. The practitioner creates their own dataset using this tool chain and handles the integration with the MSR logic of the tool as shown in Figure 1. Between researcher's and practitioner's project settings, most of the modules required changes. As we show in our adoptability evaluation experiments, for adopting *Association Mining* tool from JUnit project setting to JEdit project setting required changing four modules to remove 180 lines of code (LOC) out of 422 and add 191 LOC.

3) Customizability: Finally, if the practitioner needs to perform few customizations to the adopted tool, such as "*changing the mining logic to perform package-level association instead of file-level association*", it requires changing multiple components in the tightly integrated *Association Mining* MSR tool. As we show in our customizability evaluation experiments, this customization required changing four modules to remove 8 LOC, and add 34 LOC.

In the next section, we provide an overview of the Candoia platform and show how these challenges are addressed.

III. Candoia platform & ecosystem

We now describe the Candoia's process of building, sharing, and adopting MSR apps using our motivation scenario example and Figure 2. As shown in Figure 2, (1) the researcher will first use Candoia platform to prepare a dataset for his project (JUnit). The Candoia platform uses the in-house data extraction tools (parsers, and adapters) to read the user project and create a custom dataset. This dataset can be mined using the data abstractions of the platform. (2) The researcher then

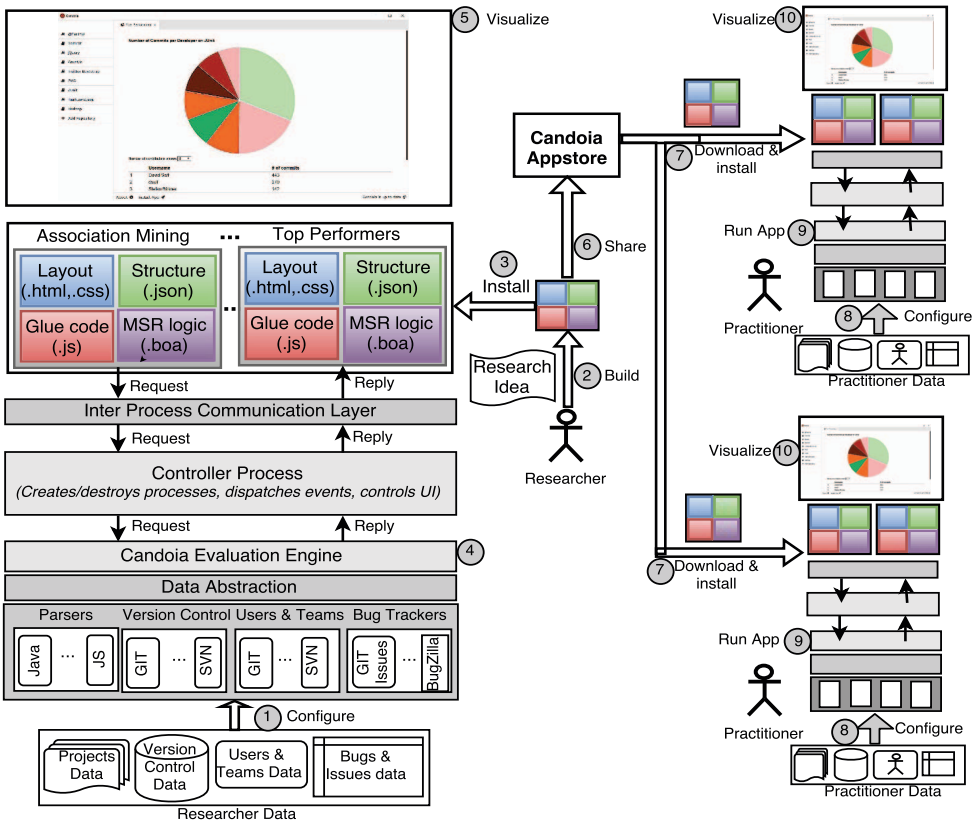

Fig. 2. Candoia platform's architecture and operational overview

builds the *Association Mining* MSR tool as an Candoia app by defining various parts of the app, such as app structure, app layout, mining logic, and glue code for binding the various components. (3) The researcher will install the app in the platform and (4) runs it using the Candoia evaluation engine. (5) The researcher can visualize the app's output and (6) share the app via Candoia appstore.

The practitioner who wants to adopt the *Association Mining* Candoia app, (7) downloads the app from the appstore and installs it in the platform. (8) The practitioner will use the platform to prepare a dataset for his project (JEdit). (9) The downloaded *Association Mining* app can be readily run and (10) output can be visualized without requiring any additional efforts. For customizing the app to *perform package-level association instead of file-level association*, the practitioner will modify the mining logic component, which does not require any changes to other components of the app. As we show in our customizability evaluation, this customization in Candoia required changing just 1 line of code to the MSR logic component. After customizing the app, the practitioner needs to simply install and run to visualize the changes.

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

1) *Applicability*: Candoia should enable building robust MSR tools by supporting the common MSR technologies and providing extension points to add new technologies. Also, it should be easier to describe various components of a Candoia app.
2) *Commonality*: Identifying the common components across MSR apps and providing them as part of the platform to make Candoia apps light-weight.
3) *Adoptability*: Adopting an app is simply by "*Install & Run*". An app built for one project setting can run across diverse project settings without requiring any change.
4) *Customizability*: Facilitate easy customization of apps by clearly defining various components of a Candoia app and choosing efficient script-based domain-specific languages (DSLs) to build the components, the idea here is that scripts are easier to customize than programs.
5) *Security*: Secure Candoia user's system against third-party Candoia apps, and secure one app from another.
6) *Scalability*: Process-level parallelism in isolation; each app runs as a process.

A. Candoia For Building Robust MSR Apps

By applicability we mean the ability of the Candoia platform to enable building of a variety of MSR tools. We explored *different MSR artifacts* used by MSR tools in the past, such as software project source code, version control data, bug data, users and teams data, mailing lists, etc, and gathered

different sources of these MSR artifacts, such as source code written in different programming languages, bug data coming from Bugzilla, JIRA repositories, GIT, SVN, or CVS version control data etc. Upon determining the variety of MSR artifacts and their sources, we built a set of data extraction tools (mainly includes language parsers, adapters to read version control data, bug data, etc) and provided them as part of Candoia platform, such that Candoia when configured using user projects can automatically extract different MSR artifacts and prepare user datasets. At present Candoia supports SVN, Git as VCS, Bugzilla, GitHub-Issues, JIRA, SF-Tickets as bug databases, SF.net, GitHub as forges and Java, Javascript as programming language. Candoia also allows users to add their own data extraction tools as long as the read data complies with Candoia's data schemas.

Now that Candoia supports common MSR technologies to build a variety of MSR tools, it is important to enable building of robust tools. We achieve this by using powerful domain-specific languages for expressing various functionalities of the app. These DSLs are reasonably well-known such that it is accessible to most developers and involve smaller learning curve. 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 [14] [15], and for writing glue code to manage interaction, updates, and data exchange in an app we selected Javascript.

B. Candoia App Structure

Building a Candoia app consists of defining four parts: the MSR logic, the structure description of the app, the layout description for the visualization and glue code. The listing below describes different components of a Candoia app.

- *package.json*: metadata about the app,
- *main.html*: describes visual layout,
- *app.css*: app's stylesheet,
- *main.js*: contains glue code for interaction,
- <*app-name*>.*boa*: MSR logic(extension of Boa DSL),
- *lib*: libraries used by this app.

The structure description of a Candoia app is described by its *package.json* configuration file. 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). For instance, Weka or R Javascript bindings can be used in the app for model building. Candoia's language for writing an app's MSR logic is an extension of the Boa language [14]. Boa is a domain-specific language specifically introduced for MSR. The interaction between the components is done only via the Candoia front-end APIs. For instance, a typical Candoia app first fires a mining query described in the file <app−name >.boa using api.boa.run(<app−name>.boa) API. The details about the secured interaction between various components of the client code (described in the app), the dataset, and the file system via chromium platform is described in §III-F.

C. Candoia Data Abstraction Layer

Candoia's data abstraction layer is the key to achieving portability (or adoptability) of Candoia apps across diverse project settings. Candoia's data abstractions hide the details about the data sources. As Candoia supports multiple forges, programming languages,VCSs, and multiple bug repositories, the abstractions provide uniform access to different MSR artifacts originating from different data sources.

Programming using higher level data abstractions was previously used by several approaches that have tried to provide uniform access to data from disparate sources. For instance, Boa [14] [15], provides data abstractions for GIT and SVN version control data, and project metadata from GitHub and SF.net forges. Defect4j [25] provides abstractions for version control data. We extend Boa's data abstractions to add new abstractions for bug repositories such as Bugzilla, JIRA, etc. We also extended the existing user abstractions with abstractions to represent team and organization data. Figure 3 provides a high level overview of our data schema [40] (highlighted text indicates the additions to Boa's data schema). Like Boa we use Protocol Buffers [1] to store MSR

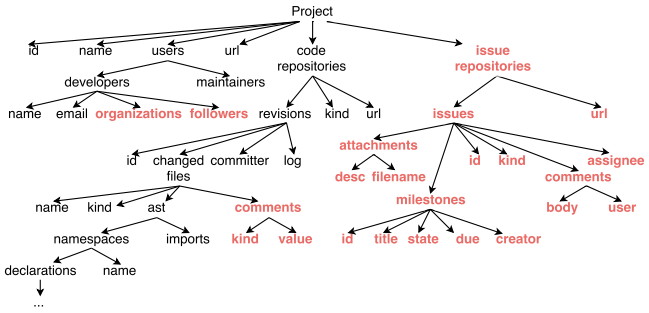

Fig. 3. Candoia's data schema [40].

artifacts. Boa language provides domain-specific types for programming using data abstractions. We have extended the set of domain-specific types provided by Boa to include new types for representing bug/issue data. We have also extended the project meta data types to include organization data along with user/committer data. In Figure 4 we show a snippet from an app's mining logic component (boa program) that uses Issue data abstractions and types to mine *bugs that will never be fixed*. The issue kind is used in the mining logic to filter these bugs.

```
1  ...
2  will-never-be-fixed-bugs : output collection of string;
3  visitor(p, visitor {
4      before issue : Issue -> {
5          if ((issue.kind == IssueKind.WONTFIX) ||
6              (issue.kind == IssueKind.INVALID) ||
7              (issue.kind == IssueKind.WORKSFORME))
8              will-never-be-fixed-bugs << issue.id;
9      }
10 ...
```

Fig. 4. A code snippet for mining bugs that will never be fixed.

A Candoia app implementing the mining logic shown in

Figure 4 can fetch the bugs of the specified kind from several bug repositories without requiring any changes to the mining logic. In summary, Candoia's data abstraction layer provides abstractions for source code, version control data, bug data, and user and organization data, and provides uniform access to data originating from several sources to achieve compatibility of apps across diverse project settings.

D. Customizability

Customizations in Candoia are of two types: i) data source customizations, and ii) app customizations. The first kind of customizations are concerned with changing the data source. For instance, using GitHub−Issues as bug repository instead of Bugzilla. The data source customizations are automatically handled by the platform and they do not require any changes in the app. The second kind of customizations are concerned with changing different parts of an app. For instance, modifying the MSR logic, changing the output format, customizing to perform post-processing of the results using weka library, etc.

The app customizations in Candoia are more focused in terms of findings the right component(s) for performing customizations and uses languages designed for that purpose. A Candoia app is well-structured into different components and the app structure not only helps to locate the component for customization, but also enforces disciplined customizations. This can also be achieved if an MSR tool not using Candoia is engineered carefully; however it requires extra work to enforce this design discipline. Every component of a Candoia app is written using a script-based domain-specific language (DSL) and often scripts are easier to customize than programs.

To give concrete examples of customizations in a Candoia app, consider the *Association Mining* app. This app predicts bug by mining file associations. The app uses the version control data, the source files, and bug data. The app's mined results are used as input to Weka's Apriori association mining algorithm to predict which files are associated with each other. We now list a number of customizations of this app and show how it is performed in Candoia.

— The app currently uses *Apriori* association algorithm and it can be customized to use *Eclat* association algorithm by simply using "*api.weka.associationEclat*" API instead of "*api.weka.apriori*" API in the JavaScript component.

— The Javascript binding used to import the *Eclat* association algorithm is currently Weka, it can be changed to use a more efficient implementation of *Eclat* in SPMF, which is an open-source data mining library. This customization is done by simply using "*api.spmf.associationEclat*" API instead of "*api.weka.associationEclat*" API in the JavaScript component.

— The app performs file-level association, but package- or module-level association can be performed by changing the underlying MSR logic (requires changing 1 line of code).

— The app finds the file associations of buggy files. A customization that considers all file associations needs to ignore the bug data while computing file associations (requires changing the mining logic to ignore the bug data).

Candoia allows easy extension of the core system by providing well defined extension points for adding different system components such as new forge, VCS or language parsers etc. For example, adding a new VCS requires to write a class, which extends AbstractConnector class and implements few abstract methods(4 in VCS case). Similar extension points are available for different components [41].

E. Candoia Evaluation Engine

Candoia evaluation engine is inspired from the query engine of Boa. Boa query engine runs on a Hadoop cluster for processing thousands of open source projects from fixed datasets. For Candoia, we needed a query engine that (1) could run on a single node, (2) is able to read and process local and remote projects, and (3) provides 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 and utilizes process and thread level parallelization for running multiple MSR apps. In a nutshell, the Candoia evaluation engine works as follows: the input to Candoia evaluation engine are: i) dataset created using user projects and ii) Boa script that describes the MSR logic. The output of the evaluation engine is the expected output of the app's MSR logic. The Candoia evaluation engine processes each project in the user dataset and applies the mining logic described in the Boa script to produce the desired output.

F. Security Architecture of the Candoia platform

A key concern for Candoia is to allow apps to communicate with the platform in a safe way, and to allow access to user's data on a need-to-know basis. We also need to prevent apps from corrupting each other. We have solved these technical challenges by building on top of the Chromium platform [42]. 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 interprocess 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.

```
1 <h2> My First \FRAMEWORKNAME{} Application </h2>
2   <div id='content'></div>
3   <script>
4     var data = api.boa.run('myprog.boa');
5     document.getElementbyId('content').
6         innerHTML(JSON.stringify(data, null, '\t'));
7   </script>
```

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 version (api.meta)
- Inter-Process-Communication 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)"
```

G. 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.

IV. Evaluation

This section presents our empirical evaluation on different aspects of Candoia in developing MSR apps: applicability, adoptability and customizability. Apps were run on a set of 10 widely-known open source projects, hereon called test projects, as shown in Figure 5. They are chosen from diverse domains and have been actively using the two most popular version control systems (VCS), Git and SVN, and 4 widely-used issue tracking systems, Bugzilla, JIRA, SourceForge and GitHub. They are written mainly in Java or JavaScript which are the two programming languages Candoia currently supports. Their sizes range from some thousands lines of code to almost a million lines of code.

Projects	VCS	PL	Bugs	#LOC	#Revs	#Bugs	#Devs
Tomcat 8.0.24 (TC)	SVN	Java	Bugzilla	381350	17433	3023	32
Hadoop 2.7.1 (HD)	Git	Java	JIRA	2217636	14301	10333	146
JUnit 4 (JU)	Git	Java	GitHub	30535	2115	148	127
SLF4j 1.7.12 (SLF)	Git	Java	JIRA	20866	1436	332	59
Bootstrap 3.3.5 (BT)	Git	JS	GitHub	65885	11840	213	718
Node.js 0.12.7 (ND)	Git	JS	GitHub	3405739	14695	955	105
Grunt 0.4.6 (GT)	Git	JS	GitHub	3596	1399	155	29
JQuery 2.1.4 (JQ)	Git	JS	GitHub	45212	6153	165	87
PMD 5.3.3 (PMD)	Git	Java	SF	175866	8736	1394	102
JEdit 5.2.0 (JE)	SVN	Java	SF	224127	24509	3926	7

Fig. 5. Test projects.

A. Applicability

Our claim is that MSR tasks and hypotheses can be expressed and evaluated using Candoia platform's capabilities. To evaluate the applicability of Candoia, we created apps for a set of MSR tasks and hypotheses that have been studied in the literature of MSR research.

Figure 6 describes our list of Candoia apps categorized into four categories: I) Bugs, II) Software Evolution, III) Project Management, and IV) Source code analysis and Programming practices.

The mining tasks in these apps analyze different kinds of MSR artifacts such as identifier names and abstract syntax trees of the source code, log messages and authors of commits in the change histories, and issues in the issue tracking systems. They analyze both general changes and bug fixing changes. Some apps were written to detect problems in programming practices (*naming convention*, *serialization-related properties*, *proper declaration of constants*), concurrency (*double checked locking*, *wait-notify features*), logic (*deeply nested if statements*), optimization (*dead code*), bad assumptions (*improper use of null*), etc.

The apps were executed on the test projects listed in Figure 5 on a machine which 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. Figure 6 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 detailed descriptions of these apps along with their source code is made available via Candoia website [9].

Results Analysis. 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 Figure 6 on test projects and discuss couple of the interesting results that our apps produced as a result. Note that, analyzing the results to draw conclusions is not our objective. Results for the apps that are not discussed here can be found in the Candoia website [9].

App #5. Identifies fixing revisions that add null checks. We found a large number of such revisions in test projects. Figure 7 shows the relative number of null checking revisions. For some projects, the frequency of these fixes is quite significant, and for others e.g. Grunt, its quite surprising to see very low number of such fixes.

App #14. Maps modules to developers. Nagappan *et al.* [32] 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: *NOA: Number of developers who contributed to the componen*, *EF: Component edit frequency*, and *DMO: Group of developers with 70% or more edits to component*. 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 which, can be related to the bugs in the project. Figure 8 shows the values for *NOE*, *EF* and *DMO* metrics along with the number of bugs in the

#	Candoia App	Number of lines of code					Execution time (s)									
		Boa	JS	HTML	CSS	JSON	TC	HD	JU	SLF	BT	ND	GT	JQ	PMD	JE
	I. Bugs															
1	Detects unreproducible or wont-fix bugs	44	48	38	33	16	30.6	110.0	5.9	2.6	40.5	149.0	2.1	10.1	20.6	47.5
2	Detects improper usage of null	45	11	25	0	16	33.0	152.0	5.8	3.5	4.8	26.3	1.1	3.3	35.8	89.4
3	Detects improper use of double checked locking idiom	100	6	25	32	16	17.0	74.0	3.3	1.6	4.2	24.4	3.0	1.1	15.0	55.4
4	Detects improper usage of wait-notify idiom	39	52	47	32	16	8.1	28.4	2.3	1.2	2.5	12.2	1.8	0.9	8.9	23.1
5	Identifies fixing revisions that add null checks	98	13	43	32	16	3.5	8.1	1.4	2.1	4.7	23.4	5.0	1.4	3.8	5.2
	II. Software Evolution															
6	Lists most frequently changed files	08	16	43	0	16	28.7	114.0	5.9	26.2	35.7	125.0	2.2	10.9	19.1	57.2
7	Lists commits that involved a large number of files	10	52	47	32	16	36.1	124.0	7.8	4.0	43.9	108.0	2.9	12.5	23.2	48.9
8	Commit blame assignment based on increase in repository size	27	52	47	32	16	60.9	163.0	9.8	4.7	62.0	189.0	3.2	19.7	32.5	89.6
9	Provides details of latest revision, e.g. total changed files etc.	10	52	47	32	16	33.0	95.1	7.0	3.1	36.9	100.0	2.6	12.2	20.2	48.12
10	Provides details of developers' last commits	55	42	41	0	16	42.7	139.0	11.8	9.1	48.1	119.0	8.25	17.7	28.4	92.7
11	Mining co-changed files via association mining	20	12	34	0	16	11.2	7.9	7.3	7.8	10.2	46.8	0.1	9.2	9.4	86.4
12	Compute churn rate for fixing bugs	13	33	47	0	16	1.5	3.7	1.4	1.0	2.6	8.6	0.5	1.1	2.8	2.2
	III. Project Management															
13	Ranks developers by the number of commits	11	52	47	32	16	31.7	111.0	5.4	2.6	42.2	137.0	2.5	11.4	22.0	46.4
14	Maps modules to developers	36	48	38	33	16	37.3	127.0	7.2	4.0	46.5	171.0	2.5	12.0	24.8	53.0
15	Computes number of attributes (NOA)	17	106	36	0	16	5.0	19.4	1.8	1.1	2.3	9.3	0.7	1.4	5.5	10.3
16	Computes number of public methods (NPM)	19	106	36	0	16	1.1	23.9	2.1	6.5	2.2	9.2	0.7	1.6	6.1	6.2
17	Identifies developers writing empty or one word commit logs	27	52	47	32	16	31.3	110.0	6.4	2.6	35.8	128.0	2.4	11.0	35.0	46.8
18	Associate bugs and source files	37	30	47	32	16	67.4	321.8	10.9	5.1	5.5	8.7	1.0	1.9	47.3	84.8
	IV. Program analysis															
19	Detects violation of naming conventions	48	48	38	33	16	10.7	37.9	0.7	1.8	2.5	18.4	1.2	0.4	15.3	22.8
20	Checks serialization-related properties	51	51	47	32	16	7.6	23.3	3.5	1.5	2.6	9.6	0.8	1.7	33	17
21	Detects static fields which are public but not final	44	48	38	33	16	7.4	28.7	2.9	1.3	2.6	10.0	0.7	1.5	9.4	15.7
22	Identifies locations of dead code	47	52	47	32	16	18.2	110.0	4.8	2.2	4.3	31.6	1.1	4.4	21.6	77
23	Identifies deeply nested if statements	25	52	47	32	16	11.9	43.6	2.9	1.4	2.6	13.9	0.9	2.0	11.5	33.9
24	Computes various popularity metrics e.g. CK, OO etc.	150	32	54	32	16	30.4	68.5	3.8	2.0	2.4	14.9	0.9	1.9	31.3	44.4

Fig. 6. Several Candoia apps with their lines of code in different languages and execution times (in seconds).

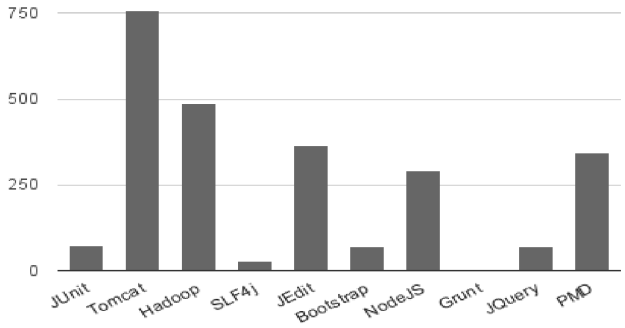

Fig. 7. Number of fixing revisions that add null checks.

projects. For quite a few projects there is a strong correlation between bugs and the EF metrics.

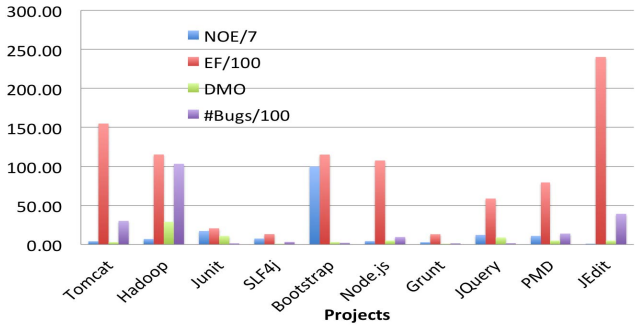

Fig. 8. Influence of organizational metrics *NOE*, *EF* and *DMO* on software quality.

B. Adoptability

In this section we show that Candoia apps are portable across diverse project settings and require no changes. For comparison purposes, we have implemented all of our MSR tasks using Java. We compare LOC changes required in both the Java version and the Candoia version for adopting apps from one project setting to another. Our collection of test projects provides us 6 different project settings as shown in Figure 10. Among 16 possible combinations of 2 VCS, 2 PLs, and 4 BTs, our test projects cover the 6 most popular ones. In the 6 project settings shown in Figure 10, setting #1 is used as our base setting (this selection is based on the popularity). Building apps in Java requires reading the project forge data, version control data, bug data, etc. We have designed these Java apps for change. Apps contain 5 modules: M_{VCS} for reading version control data, M_{Forge} for downloading project and its meta data, M_{Bug} for reading bug or issue data, M_{Mining} contains the actual mining code, and $M_{Visualize}$ module contains visualization related code. This strategy is adopted so that the design decisions that are likely to change are hidden within each modules [34]. For instance, if we have to change an app to read SVN data, instead of GIT data, we plug-in a different VCS module and rest of the code requires no change. There is a threat that, by modularizing the code, we may add few extra LOC, however we tried to keep this effect minimal. Candoia apps code is distributed among four components: Boa mining code, JS glue code, HTML and CSS code.

Results. Figure 9 compares LOC changes required for adopting apps from one project setting to another in Java and Candoia versions. The table shows comparison results for four apps, comparison result for other apps can be found in our website [9]. For each app, there are 6 rows, where

	#	Java						Candoia				
		M_{VCS}	M_{Bug}	M_{Forge}	M_{Mining}	$M_{Visualize}$	Total	Boa	JS	HTML	CSS	Total
Nullcheck	1	125	157	20	143	53	498	59	12	34	0	105
	2	148 (-89,+112)	117 (-119,+79)	27 (-15,+22)	156 (-43,+60)	53 (-1,+1)	501 (-267,+274)	59	12	34	0	105
	3	125 (-2,+2)	129 (-110,+82)	20 (-1,+1)	155 (-21,+33)	53 (-1,+1)	482 (-135,+118)	59	12	34	0	105
	4	125 (-2,+2)	115 (-111,+69)	20 (-1,+1)	167 (-18,+42)	53 (-1,+1)	480 (-133,+115)	59	12	34	0	105
	5	148 (-89,+112)	116 (-110,+69)	27 (-15,+22)	154 (-48,+59)	53 (-1,+1)	498 (-263,+263)	59	12	34	0	105
	6	120 (-15,+10)	157 (-1,+1)	20 (-1,+1)	147 (-13,+17)	53 (-1,+1)	497 (-31,+30)	59	12	34	0	105
File Association	1	72	139	20	138	53	422	20	12	34	0	66
	2	125 (-38,+91)	60 (-113,+34)	27 (-15,+22)	140 (-45,+47)	53 (-1,+1)	405 (-212,+195)	20	12	34	0	66
	3	72 (-1,+1)	146 (-120,+127)	20 (-1,+1)	146 (-7,+15)	53 (-1,+1)	437 (-130,+145)	20	12	34	0	66
	4	72 (-1,+1)	115 (-106,+72)	20 (-1,+1)	137 (-4,+3)	53 (-1,+1)	397 (-113,+78)	20	12	34	0	66
	5	125 (-38,+91)	95 (-96,+52)	27 (-15,+22)	133 (-30,+25)	53 (-1,+1)	433 (-180,+191)	20	12	34	0	66
	6	72 (-1,+1)	139 (-1,+1)	20 (-1,+1)	138 (-1,+1)	53 (-1,+1)	421 (-5,+5)	20	12	34	0	66
Churn Rate	1	52	0	20	69	53	194	13	33	47	0	93
	2	104 (-38,+90)	0	27 (-15,+22)	74 (-26,+31)	53 (-1,+1)	258 (-80,+144)	13	33	47	0	93
	3	52	0	20 (-1,+1)	69	53 (-1,+1)	194 (-2,+2)	13	33	47	0	93
	4	52	0	20 (-1,+1)	69	53 (-1,+1)	194 (-2,+2)	13	33	47	0	93
	5	104 (-38,+90)	0	27 (-15,+22)	74 (-26,+31)	53 (-1,+1)	258 (-80,+144)	13	33	47	0	93
	6	52	0	20 (-1,+1)	69	53 (-1,+1)	194 (-2,+2)	13	33	47	0	93
BugSrc Mapper	1	78	152	20	73	53	376	37	30	47	32	146
	2	105 (-49,+76)	79 (-118,+45)	27 (-15,+22)	74 (-41,+42)	53 (-1,+1)	338 (-224,+186)	37	30	47	32	146
	3	78 (-2,+2)	104 (-111,+63)	20 (-1,+1)	78 (-28,+33)	53 (-1,+1)	333 (-143,+100)	37	30	47	32	146
	4	78 (-2,+2)	85 (-106,+39)	20 (-1,+1)	77 (-24,+28)	53 (-1,+1)	313 (-134,+71)	37	30	47	32	146
	5	108 (-44,+74)	85 (-106,+39)	27 (-15,+22)	69 (-45,+41)	53 (-1,+1)	342 (-211,+177)	37	30	47	32	146
	6	78 (-2,+2)	152 (-1,+1)	20 (-1,+1)	78 (-28,+33)	53 (-1,+1)	381 (-33,+38)	37	30	47	32	146

Fig. 9. Compares LOC changes required for adopting apps from one project setting to another in Java and Candoia.

#	VCS	PL	Bugs
1	GIT	Java	Issues
2	SVN	Java	Bugzilla
3	GIT	Java	JIRA
4	GIT	Java	Tickets
5	SVN	Java	Tickets
6	GIT	JS	Issues

Fig. 10. Six project settings

the first row shows the LOC for our base project setting, and the other 5 rows shows the LOC changes required for adopting the app from base setting to another. For instance, for Nullcheck app, #1 is our base setting and the Java M_{VCS} module requires 125 LOC. For the same app, #2 is another project setting and the Java module M_{VCS} requires 148 LOC, where adopting this module from base setting required us to remove 89 LOC and add 112 LOC. For some modules we see 0 LOC, indicating that the app does not use that module. All the modules in the Candoia platform required no changes in terms of LOC, this is mainly because Candoia apps are implemented on data abstractions and not on raw data. Being able to run all Candoia apps on 6 different project settings without requiring any changes shows that apps built on Candoia platform are portable across project settings. It can also be seen that, MSR apps built on other platforms such as Java, requires considerable amount of changes (in terms of LOC) for making them portable across project settings. For instance, in Java platform for adopting Nullcheck app that is originally implemented for project setting #1 to project setting #2, required a total of 267 lines to be deleted and 274 lines to be added.

C. Customizability

We evaluate our claim that performing customizations in Candoia requires less efforts in terms of LOC. Like our adoptability evaluation, we have implemented all of our customizations in Java and Candoia and we compare the customization efforts. We compare LOC changes required in both Java and Candoia as a proxy measure of customization efforts. We report data on same four apps as in our adoptability evaluation for this evaluation, and we have listed a number of app-specific customizations for each of these four apps (we ignore the data source specific customizations, because they are already covered in our adoptability evaluation). Figure 11 lists our results for four apps and the results for other apps can be found in Candoia website [9].

From the variety of customization tasks spanning across four apps, it can be seen that for most customizations, Candoia required less number of LOC changes, except for UI related customizations. The less LOC requirement of the customizations in Candoia is mainly due to script-based DSLs that were used to write the components. In case of UI related customizations, for instance, consider the row for c_{41}, where Java required (-8,+8) LOC changes, whereas Candoia required (-28,+35). This was mainly due to the difference in the visualization library that is used in Java and Candoia. In the Java implementation, we used google charting library which is designed to be adaptable, whereas in Candoia we used the standard JavaScript chart.js. From the results we can also observe that customizations in Java requires changing every module, whereas customizations in Candoia requires changing fewer number of modules (more focused customization). One

c_{10}	Shows number of nullcheck bug revisions in pie chart	c_{23}	Module association instead of file association
c_{11}	Change the output display to column chart	c_{24}	File association without bug data
c_{12}	Display nullcheck issue life time	c_{30}	Churn rate based on revisions
c_{13}	Plot nullcheck date v/s number of modified files	c_{31}	Associate bugs to churn rates
c_{14}	Maps nullcheck to developers	c_{40}	Bugs to source files mapping displayed in column chart
c_{20}	File associations using weka apriori	c_{41}	Change the output display to pie chart
c_{21}	File associations using weka fpgrowth	c_{42}	Top five files with maximum bug fix time
c_{22}	File associations using spmf eclat	c_{43}	Asssociate developers to bugs

	#	Java						Candoia				
		M_{VCS}	M_{Bug}	M_{Forge}	M_{Mining}	$M_{Visualize}$	Total	Boa	JS	HTML	CSS	Total
Nullcheck	c_{10}	125	157	20	143	53	498	59	41	45	26	171
	c_{11}	125 (-1,+1)	157 (-1,+1)	20 (-1,+1)	143 (-2,+2)	53 (-3,+3)	498 (-8,+8)	59	12	34	0	105
	c_{12}	125 (-1,+1)	137 (-29,+9)	20 (-1,+1)	144 (-14,+11)	53 (-2,+2)	479 (-47,+24)	74 (-4,+19)	41 (-2,+2)	45 (-4,+4)	26 (-1,+1)	186 (-11,+26)
	c_{13}	125 (-1,+1)	157 (-1,+1)	20 (-1,+1)	147 (-6,+11)	53 (-1,+1)	501 (-10,+15)	64 (-3,+8)	41 (-4,+4)	45 (-4,+4)	26 (-1,+1)	176 (-12,+17)
	c_{14}	125 (-1,+1)	157 (-1,+1)	20 (-1,+1)	147 (-13,+18)	53 (-1,+1)	502 (-17,+22)	61 (-4,+1)	41 (-4,+4)	45 (-4,+4)	26 (-1,+1)	173 (-13,+10)
File Assoc.	c_{20}	141	157	20	178	23	481	37	12	34	0	83
	c_{21}	141 (-1,+1)	157 (-1,+1)	20 (-1,+1)	178 (-3,+3)	23 (-1,+1)	481 (-7,+7)	37	12 (-1,+1)	34	0	83 (-1,+1)
	c_{22}	141 (-1,+1)	157 (-1,+1)	20 (-1,+1)	183 (-23,+28)	23 (-1,+1)	486 (-27,+32)	37	12 (-1,+1)	34	0	83 (-1,+1)
	c_{23}	141 (-1,+1)	157 (-1,+1)	20 (-1,+1)	178 (-3,+3)	23 (-1,+1)	461 (-8,+34)	37	12 (-1,+1)	34	0	83 (-1,+1)
	c_{24}	141 (-1,+1)	0	20 (-1,+1)	175 (-5,+2)	23 (-1,+1)	359 (-165,+5)	24 (-20,+7)	12 (-1,+1)	34	0	70 (-21,+8)
Churn	c_{30}	52	0	20	69	53	194	13	33	47	0	93
	c_{31}	72 (-1,+21)	0	20 (-1,+1)	73 (-4,+8)	53 (-1,+1)	218 (-7,+31)	42 (-4,33)	33	47	0	122 (-4,+33)
BugSrc	c_{40}	78	152	20	73	53	376	37	30	47	32	146
	c_{41}	78 (-2,+2)	152 (-2,+2)	20 (-1,+1)	73 (-1,+1)	53 (-2,+2)	376 (-8,+8)	37	38 (-28,+35)	47	32	154 (-28,+35)
	c_{42}	78 (-2,+2)	152 (-2,+2)	20 (-1,+1)	137 (-18,+82)	53 (-1,+1)	440 (-24,+88)	41 (-15,+19)	30	47	32	155 (-15,+19)
	c_{43}	78 (-2,+2)	157 (-17,+23)	20 (-1,+1)	99 (-19,+47)	53 (-1,+1)	407 (-40,+74)	46 (-2,+11)	38 (-4,+12)	47	32	163 (-6,+23)

Fig. 11. Compares LOC changes required for a number of customizations in Java and Candoia.

could argue that the modularization strategy for Java apps is the reason behind this, however we did not change the modularization strategy for individual evaluations and we used standard strategy for modularizing the Java apps [34].

We also claim that customizations in Candoia are more focused in terms of finding the right component to change and perform the change fairly quickly. For evaluating this claim we performed a user study as described below.

Methodology. We gathered a group of eight Candoia app developers with varying expertise (excludes authors and developers of the apps used in the paper). We determine the developer expertise by asking background questions shown in Figure 12 (B1-B4). We then asked the developers to select a customization task and their preferred project setting from the list of customization tasks and project settings shown in Figure 12. Each developer performs the following tasks (in order): 1) answers a questionnaire about their background, 2) selects a Candoia app and a project setting from the list of project settings, 3) runs the Candoia app on the selected project setting, 4) customizes the Candoia app based on the customization requirement provided to selected app, 5) re-runs the customized app on the previously selected project setting, 6) also runs the customized app on a new project setting, 7) answers another questionnaire at the end of the task.

Results. We recorded developer responses to background questionnaire and Candoia experience questionnaire. We also recorded the time they took to complete the customization task. Figure 12 shows the recorded responses.

From Table 12, 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 1 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.

D. Threats To Validity

Threats to internal validity concern our selection of test projects and apps for evaluation. To mitigate test projects threat, we have selected only open source projects that are widely used, actively maintained and have been used in the past for evaluating MSR techniques. To mitigate bias in the selection of apps, we have selected apps spanning into multiple categories. We have also included a number of apps that fully/partially implements the MSR tools/techniques published in previous years of MSR conferences.

Threats to external validity concern the possibility to generalize our results, i.e. can Candoia be used in other settings than tested settings? Candoia currently supports Java and Javascript programming languages, GIT, SVN and CVS version controlling, Bugzilla, JIRA, GitHub−Issues and SF−tickets for bug data, and GitHub and SF.net for project metadata and user and organization data. Supporting other languages may be challenging, such as C/C++ which offers language features that differs significantly from Java/Javascript. We do

#	Task Description			
1	*App #1*: Include duplicate bug reports			
2	*App #6*: Apply year filter 2010			
3	*App #15*: Display the trend over revisions			
#	Project	VCS	PL	Bug
1	Bootstrap	Git	JS	Issues
2	JUnit	Git	Java	Issues
3	Tomcat	SVN	Java	BugZilla

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

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

Dev	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

Fig. 12. List of customization tasks and project settings used in the study. Responses recorded from eight developers, where B1-B4 records the developers background and E1-E3 records developers experience with Candoia .

not see problems supporting other non-commercial forges, VCS, and bug repositories. Commercial repositories are not tested, however they can be easily supported, as they don't differ much from the popular open-source repositories.

V. Related Work

Our idea of a platform and an ecosystem for building and distributing MSR tools is novel; however, we draw inspiration from a rich body of work in this area. In terms of its focus, the Candoia platform is closer to the Moose platform [13], RepoGrams [39], Kenyon [6], Sourcerer [3], Alitheia Core [19], [20], FLOSSMole [24] and different from PROMISE Repository [37], Open-access data repositories [17], Black Duck OpenHub (aka Ohloh) [8], GHTorrent [18], [21], SourcererDB [33], Boa [14] [15], and the SourceForge Research Data Archive (SRDA) [16]. The former set of approaches provide frameworks for building tools, 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 [31]. We had presented an earlier version of this work in a poster paper [44].

Moose is a platforms for reusing of data mining tools and allow low cost addition of new tools. The main difference is in terms of focus. Candoia is focused on MSR apps so it integrates support for VCS, bug tracking, etc, which isn't easily available in Moose. RepoGrams [39] is a tool for comparing and contrasting of source code repositories of software projects with respect to a set of metrics. Candoia and RepoGrams both consume source code repositories of software projects, and both Candoia apps and RepoGrams metrics can be used to analyze the source code repositories. The key difference is in the purpose; RepoGrams helps researchers gather evaluation targets for evaluating a research prototype, while Candoia is used to build the research prototype that is compatible across diverse project settings. Both Kenyon and Sourcerer define database schemas for metadata and source code, and provide access to this dataset via SQL. 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. Similarly, FLOSSMole gathers metadata (e.g., project topics, activity, statistics, licenses, developer skills etc) and allows analysis on them. Groundhog [35] is an infrastructure for downloading projects from SourceForge, analyzing the source code, and collect metrics from these projects. When compared to these approaches, Candoia provides data abstractions for several MSR artifacts such as project metadata, revisions, source code, bugs, users and teams, that originates from multiple sources. This aspect of Candoia make the apps built on top of data abstractions compatible across diverse project settings.

GHTorrent, PROMISE Repository, SourcererDB, and Boa provide a repository of datasets from open-source projects so that researchers do not have to collect and curate datasets. When compared to these set of approaches that are focused on providing standard datasets, Candoia allows mining of user specific datasets. Also, Candoia allows mining of a variety of MSR artifacts. SourcererDB on top of providing datasets also provides a framework for users to create custom datasets using their projects. SourcererDB's future work presents number of challenges that are addressed in Candoia. Boa also provides an infrastructure for mining the fine grained program elements of the source code and revision history but on a very large and fixed datset from open source repositories. Candoia provides facilities to analyze user's private projects.

VI. Conclusion and Future Work

In this work, we present Candoia, a platform and an ecosystem to ease building and sharing MSR tools, where MSR tools are built as apps and Candoia platform handles the portability, and customizability aspects of apps. The Candoia ecosystem, acting as an appstore, enables sharing of apps. We have implemented both the Candoia platform and the Candoia ecosystem and evaluated by building over two dozen apps in four different categories. Our evaluation demonstrates that Candoia can be used to build a variety of robust MSR apps that are portable across diverse project settings. Furthermore, customizations of Candoia apps to suit user'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 are very excited about new apps that us and others can develop for the platform.

VII. Acknowledgement

This work was supported in part by the US National Science Foundation under grants CCF-15-18897, CNS-15-13263, and CCF-14-23370. The authors would also like to thank Dalton D. Mills and Trey Erenberger for helping with Candoia frontend implementation, Eric Lin for implementing several Candoia apps, Ramanathan Ramu for help on implementing Candoia exchange.

REFERENCES

[1] Protocol buffers. https://developers.google.com/protocol-buffers/.

[2] 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.

[3] 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.

[4] 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.

[5] J. Bevan, J. E. James Whitehead, S. Kim, and M. Godfrey. Facilitating software evolution research with kenyon. pages 177–186. 2005.

[6] 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.

[7] 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.

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

[9] Candoia website. http://candoia.github.io.

[10] 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.

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

[12] 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.

[13] S. Ducasse, T. Gîrba, and O. Nierstrasz. Moose: An Agile Reengineering Environment. *ESEC/FSE-13*, pages 99–102. 2005.

[14] 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.

[15] R. Dyer, H. A. Nguyen, H. Rajan, and T. N. Nguyen. Boa: Ultra-large-scale software repository and source-code mining. *ACM Trans. Softw. Eng. Methodol.*, 25(1):7:1–7:34, Dec. 2015.

[16] 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.

[17] 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.

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

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

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

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

[22] G. Gousios, B. Vasilescu, A. Serebrenik, and A. Zaidman. Lean GHTorrent: GitHub Data on Demand. *MSR'14*, pages 384–387. 2014.

[23] 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.

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

[25] R. Just, D. Jalali, and M. D. Ernst. Defects4J: A database of existing faults to enable controlled testing studies for Java programs. pages 437–440. 2014.

[26] 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.

[27] 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.

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

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] D. L. Parnas. On the Criteria to Be Used in Decomposing Systems into Modules. *Commun. ACM*, 15(12):1053–1058, Dec. 1972.

[35] 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.

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

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

[38] G. Robles. Replicating MSR: A study of the potential replicability of papers published in the Mining Software Repositories proceedings. pages 171–180, 2010.

[39] D. Rozenberg, I. Beschastnikh, F. Kosmale, V. Poser, H. Becker, M. Palyart, and G. C. Murphy. Comparing Repositories Visually with Repograms. *MSR'16*.

[40] The Candoia Project. Candoia: Domain Specific Types. http://candoia.github.io/docs/dsl-types.html.

[41] The Candoia Project. Candoia: Source code. https://github.com/candoia/candoia.

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

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

[44] N. M. Tiwari, G. Upadhyaya, and H. Rajan. Candoia: A platform and ecosystem for mining software repositories tools. *ICSE '16*, pages 759–764, New York, NY, USA, 2016.

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

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

[47] 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.

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