Mining Source Code Repositories with Boa

Robert Dyer, Hoan Nguyen, Hridesh Rajan, and Tien Nguyen
{rdyer,hoan,hridesh,tien}@iastate.edu

Iowa State University

The research and educational activities described in this talk were supported in part by the US National Science Foundation (NSF) under grants CCF-13-49153, CCF-13-20578, TWC-12-23828, CCF-11-17937, CCF-10-17334, and CCF-10-18600.
Why mine software repositories?

Learn from the past

Keep doing what works

Empirical validation

To find better designs

Spot (anti-)patterns

What is actually practiced

Inform the future

Learn from the past  Inform the future
Open source repositories

- Google code
- CodePlex
- Apache
- Github
- SourceForge.net
- Atlassian
- Bitbucket
- Launchpad
Open source repositories

1,000,000+ projects

1,000,000,000+ lines of code

10,000,000+ revisions

3,000,000+ issue reports
Open source repositories

1,000,000+ projects

What is the most used PL?

1,000,000,000+ lines of code

How many methods are named "test"?

10,000,000+ revisions

How many words are in log messages?

3,000,000+ issue reports

How many issue reports have duplicates?
Consider a task to answer

"How many bug fixes add checks for null?"
Has repository?

Yes

Access repository

mine revisions

Fixes bug?

Yes

Find all Java source files

mine source code

Find null checks in each source

Output count of all null checks

foreach project

Find null checks in each source

mine project metadata
A solution in Java...

class AddNullCheck {
    static void main(String[] args) {
        ... /* create and submit a Hadoop job */
    }

    static class AddNullCheckMapper extends Mapper<Text, BytesWritable, Text, LongWritable> {
        static class DefaultVisitor {
            ... /* define default tree traversal */
        }
        void map(Text key, BytesWritable value, Context context) {
            final Project p = ... /* read from input */
            new DefaultVisitor() {
                boolean preVisit(Expression e) {
                    if (e.kind == ExpressionKind.EQ || e.kind == ExpressionKind.NEQ)
                        for (Expression exp : e.expressions)
                            if (exp.kind == ExpressionKind.LITERAL && exp.literal.equals("null")) {
                                context.write(new Text("count"), new LongWritable(1));
                                break;
                            }
                }
                }.visit(p);
        }
    }

    static class AddNullCheckReducer extends Reducer<Text, LongWritable, Text, LongWritable> {
        void reduce(Text key, Iterable<LongWritable> vals, Context context) {
            int sum = 0;
            for (LongWritable value : vals)
                sum += value.get();
            context.write(key, new LongWritable(sum));
        }
    }
}

Full program
over 140 lines of code

Uses JSON, SVN, and Eclipse JDT libraries

Uses Hadoop framework

Explicit/manual parallelization
The Boa language and data-intensive infrastructure

http://boa.cs.iastate.edu/
Design goals

- Easy to use
- Scalable and efficient
- Reproducible research results
Design goals

Easy to use

- Simple language
- No need to know details of
  - Software repository mining
  - Data parallelization
Design goals

- Study *millions* of projects
- Results in minutes, not days
Design goals

Reproducible research results

Robles, MSR'10

Studied 171 papers

Only 2 were "replication friendly"

Replicating MSR:
A study of the potential replicability of papers published in the Mining Software Repositories Proceedings

Gregorio Robles
GI/CC/UnivCaM
Universidad Rey Juan Carlos
Madrid, Spain
Email: greg@cs.uji.es

Abstract—This paper is part of the result of reviewing all papers published in the proceedings of the former International Workshop on Mining Software Repositories (MSR) (2006-2008) and now Working Conference on MSR (2007-2009). We have analyzed the papers that contained any experimental analysis of software projects for their potential of being replicated. In this regard, these main issues have been addressed: i) the public availability of the data used in the study, ii) the public availability of the preprocessed datasets used by researchers and iii) the public availability of the tools and scripts. A total number of 171 papers have been analyzed from the six workshops/working conferences up to date. Results show that MSR authors use in general publicly available data sources, mainly from free software repositories, but that the amount of publicly available preprocessed datasets is very low. Regarding tools and scripts, for a majority of papers we have not been able to find any tool, even for papers where the authors explicitly state that they have built one. Lessons learned from the experience of reviewing the whole MSR literature and some potential solutions to lower the barriers of replicability are finally presented and discussed.

Keywords—replication, tools, public datasets, mining software repositories

Replication package: http://gyc.cs.uji.es/~gros/msr2010

I. INTRODUCTION

Mining software repositories (MSR) has become a fundamental area of research for the Software Engineering discipline. Among these threats, we may encounter: lack of independent validation of the presented results; changes in practices, tools or methodologies; or generalization of knowledge although a limited amount of case studies have been performed.

A simple taxonomy of replication studies provides us with two main groups: exact replications and conceptual replications. The former ones are those in “which the procedures of an experiment are followed as closely as possible to determine whether the same results can be obtained”, while the latter ones are those “in which the same research question or hypothesis is evaluated by using a different experimental procedure, i.e. many or all of the variables described above are changed” [2]. In this paper, we will target exact replications in the requirements that have to be met to perform an exact replication are more severe, and in general make a conceptual replication feasible.

We are focusing on this paper on potential replication as we have actually not replicated any of the studies presented in the papers under review. Our aim in this sense is more humble: we want to check if the necessary conditions that make a replication possible are met.

The rest of the paper is structured as follows: in the next section, the method used for this study is presented. Then some general remarks on the MSR conference is given. In the next section we analyze the papers according to the main problems presented. Finally, we provide conclusions and recommendations.
Boa architecture

Boa Language
- MapReduce
- Domain-specific Types
- Visitors

Boa's Compiler
- MapReduce
- Domain-specific Types
- Visitors
- Cached Data input reader
- User Functions
- Quantifiers
- Visitors

Boa's Data Infrastructure
- SF.net
- Replicator
- Caching Translator
- Local Cache

Recall: A solution in Java...

class AddNullCheck {
    static void main(String[] args) {
        /* create and submit a Hadoop job */
    }
}

static class AddNullCheckMapper extends Mapper<Text, BytesWritable, Text, LongWritable> {
    static class DefaultVisitor {
        /* define default tree traversal */
    }
    void map(Text key, BytesWritable value, Context context) {
        final Project p = /* read from input */
        new DefaultVisitor() {
            boolean preVisit(Expression e) {
                if (e.kind == ExpressionKind.EQ || e.kind == ExpressionKind.NEQ)
                    for (Expression exp : e.expressions)
                        if (exp.kind == ExpressionKind.LITERAL && exp.literal.equals("null")) {
                            context.write(new Text("count"), new LongWritable(1));
                            break;
                        }
                }
            }
        }.visit(p);
    }
}

static class AddNullCheckReducer extends Reducer<Text, LongWritable, Text, LongWritable> {
    void reduce(Text key, Iterable<LongWritable> vals, Context context) {
        int sum = 0;
        for (LongWritable value : vals)
            sum += value.get();
        context.write(key, new LongWritable(sum));
    }
}

Full program
over 140 lines of code

Uses JSON, SVN, and Eclipse JDT libraries

Uses Hadoop framework

Explicit/manual parallelization

Too much code!
Do not read!
A better solution...

```
p: Project = input;
count: output sum of int;

visit(p, visitor {
    before e: Expression ->
        if (e.kind == ExpressionKind.EQ || e.kind == ExpressionKind.NEQ)
            exists (i: int; isliteral(e.expressions[i], "null"))
                count << 1;
});
```

Full program **8 lines of code!**

Automatically parallelized!

**No external libraries** needed!

Analyzes **28.8 million** source files in about **15 minutes**!

(only 32 **microseconds** each!)
p: Project = input;
count: output sum of int;

visit(p, visitor {
    before e: Expression ->
        if (e.kind == ExpressionKind.EQ || e.kind == ExpressionKind.NEQ)
            exists (i: int; isliteral(e.expressions[i], "null"))
                count << 1;
        count << 1;
    count << 1;
});

count[] = 120789791
Design goals

Easy to use

Scalable and efficient

Reproducible research results
Let's see it in action!

http://boa.cs.iastate.edu/boa/
Why are we waiting for results?

Program is analyzing...

699,331 projects
494,158 repositories
15,063,073 revisions
69,863,970 files
18,651,043,238 AST nodes
Let's check the results!

<<demo>>
Domain-specific types

http://boa.cs.iastate.edu/docs/dsl-types.php

p: Project = input;
count: output sum of int;

visit(p, visitor {
  before e: Expression ->
  if (e.kind == ExpressionKind.EQ || e.kind == ExpressionKind.NEQ)
    exists (i: int; isliteral(e.expressions[i], "null"))
      count << 1;
});

Abstracts details of how to mine software repositories
Domain-specific types
http://boa.cs.iastate.edu/docs/dsl-types.php

Project

id : string
name : string
description : string
homepage_url : string
programming_languages : array of string
licenses : array of string
maintainers : array of Person

code_repositories : array of CodeRepository
Domain-specific types

http://boa.cs.iastate.edu/docs/dsl-types.php

CodeRepository

- url : string
- kind : RepositoryKind
- revisions : array of Revision

Revision

- id : int
- author : Person
- committer : Person
- commit_date : time
- log : string
- files : array of File

File

- name : string
- kind : FileKind
- change : ChangeKind
Domain-specific functions

http://boa.cs.iastate.edu/docs/dsl-functions.php

```plaintext
hasfiletype := function (rev: Revision, ext: string) : bool {
    exists (i: int; match(format(`\.%s$`, ext), rev.files[i].name))
    return true;
    return false;
};
```

Mines a revision to see if it contains any files of the type specified.
Domain-specific functions

http://boa.cs.iastate.edu/docs/dsl-functions.php

isfixingrevision := function (log: string) : bool {
  if (match(`\bfix(s|es|ing|ed)?\b`, log)) return true;
  if (match(`\b(error|bug|issue)(s)`(s)\b`, log)) return true;
  return false;
}

Mines a revision log to see if it fixed a bug.
User-defined functions

http://boa.cs.iastate.edu/docs/user-functions.php

```
id := function (a_1: t_1, ..., a_n: t_n) [: ret] {
    ... # body
    [return ...;]
};
```

Return type is optional

- Allows for complex algorithms and code re-use
- Users can provide their own mining algorithms
Quantifiers

foreach (i: int; condition...) 
    body;

For each value of i,

    if condition holds
    then
        run body (with i bound to the value)
exists (i: int; condition...)  
  body;

For some value of i,

if condition holds 
  then 
run body once (with i bound to the value)
Quantifiers
http://boa.cs.iastate.edu/docs/quantifiers.php

ifall (i: int; condition...)  
    body;

For all values of i,

if condition holds
    then
run body once (with i not bound)
Output and aggregation

http://boa.cs.iastate.edu/docs/aggregators.php

- Output defined in terms of predefined data aggregators
  - sum, set, mean, maximum, minimum, etc
- Values sent to output aggregation variables
- Output can be indexed

```java
p: Project = input;
count: output sum of int;

visit(p, visitor {
    before e: Expression ->
        if (e.kind == ExpressionKind.EQ || e.kind == ExpressionKind.NEQ)
            exists (i: int; isliteral(e.expressions[i], "null"))
            count << 1;
});
```
Declarative Visitors in Boa

http://boa.cs.iastate.edu/
Basic Syntax

id := visitor {
    before id:T -> statement
    after id:T -> statement
    ...
};

visit(startNode, id);

Execute statement either before or after visiting the children of a node of type T
Depth-First Traversal

Provides a default, depth-first traversal strategy

A -> B -> C -> D -> E

before A -> statement
before B -> statement
before C -> statement
after C -> statement
before D -> statement
after D -> statement
after B -> statement
before E -> statement
after E -> statement
after A -> statement
Type Lists and Wildcards

```java
visitor {
    before id:T -> statement
    after T2,T3,T4 -> statement
    after _ -> statement
}
```

Single type (with identifier)

Attributes of the node available via identifier
Type Lists and Wildcards

visitor {
    before id:T -> statement
    after T2,T3,T4 -> statement
    after _ -> statement
}

Type list (no identifier)

Executes **statement** when visiting nodes of type **T2**, **T3**, or **T4**
Type Lists and Wildcards

```java
visitor {
    before id:T -> statement
    after  T2,T3,T4 -> statement
    after _ -> statement
}
```

Wildcard (no identifier)

Executes `statement` for any node not already listed in another similar clause (e.g., T but not T2/T3/T4)

Provides `default` behavior
Type Lists and Wildcards

```plaintext
visitor {
    before id:T -> statement
    after T2,T3,T4 -> statement
    after _ -> statement
}
```

Types can be matched by at most 1 before clause
and at most 1 after clause
Custom Traversals

A -> E -> B -> C -> D

before n: A -> {
    visit(n.E);
    visit(n.B);
    stop;
}
Design goals

Easy to use

Scalable and efficient

Reproducible research results
Efficient execution
Efficient execution

<table>
<thead>
<tr>
<th>Number of Projects (7k, 70k, 700k)</th>
<th>Total time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.3</td>
<td>6,998</td>
</tr>
<tr>
<td>B.6</td>
<td>10,750</td>
</tr>
<tr>
<td>C.1</td>
<td>474</td>
</tr>
<tr>
<td>D.5</td>
<td>598</td>
</tr>
</tbody>
</table>

- Java
- Boa
Scalability of input size
Scalability of input size
Scales to more cores
Design goals

- Easy to use
- Scalable and efficient
- Reproducible research results
Reproducing MSR results

2/154 experimental papers "replication friendly."

48 due to lack of published data
Prior research results are difficult (or impossible) to reproduce.

Boa makes this easier!
Controlled Experiment

- Published artifacts (Boa website):
  - Boa source code
  - Dataset used (timestamp of data)
  - Results

<table>
<thead>
<tr>
<th>Expert</th>
<th>Education</th>
<th>Intro Time</th>
<th>Task 1 Task</th>
<th>Task 1 Time</th>
<th>Task 2 Task</th>
<th>Task 2 Time</th>
<th>Task 3 Task</th>
<th>Task 3 Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>Post-doc</td>
<td>6</td>
<td>B.1</td>
<td>1</td>
<td>B.6</td>
<td>4</td>
<td>B.9</td>
<td>3</td>
</tr>
<tr>
<td>Yes</td>
<td>PhD</td>
<td>5</td>
<td>A.1</td>
<td>3</td>
<td>B.6</td>
<td>2</td>
<td>B.7</td>
<td>6</td>
</tr>
<tr>
<td>No</td>
<td>PhD</td>
<td>4</td>
<td>B.6</td>
<td>1</td>
<td>B.10</td>
<td>4</td>
<td>B.9</td>
<td>4</td>
</tr>
<tr>
<td>No</td>
<td>PhD</td>
<td>4</td>
<td>A.2</td>
<td>2</td>
<td>B.6</td>
<td>2</td>
<td>D.5</td>
<td>4</td>
</tr>
<tr>
<td>No</td>
<td>MS</td>
<td>4</td>
<td>A.1</td>
<td>4</td>
<td>B.6</td>
<td>1</td>
<td>D.3</td>
<td>2</td>
</tr>
<tr>
<td>No</td>
<td>MS</td>
<td>3</td>
<td>B.6</td>
<td>2</td>
<td>C.1</td>
<td>2</td>
<td>D.4</td>
<td>10</td>
</tr>
<tr>
<td>No</td>
<td>MS</td>
<td>6</td>
<td>A.1</td>
<td>2</td>
<td>B.7</td>
<td>3</td>
<td>B.10</td>
<td>3</td>
</tr>
<tr>
<td>No</td>
<td>BS</td>
<td>2</td>
<td>A.2</td>
<td>2</td>
<td>D.1</td>
<td>2</td>
<td>D.3</td>
<td>2</td>
</tr>
</tbody>
</table>

Fig. 16. Study results. All times given in minutes.
Ongoing work

Language abstractions

Infrastructure improvements

Other artifacts

Google Code

GitHub

Launchpad

Other artifacts

Language abstractions

Infrastructure improvements

Other artifacts
Domain-specific language and infrastructure for software repository mining that is:

- Easy to use
- Efficient and scalable
- Amenable to reproducing prior results