Performing Large-scale Mining Studies, From Start to Finish

Robert Dyer
Samuel W. Flint

The research and educational activities described in this talk were supported in part by the US National Science Foundation (NSF) under grants CCF-19-34884, CNS-15-13263, CNS-15-12947, CCF-14-23370, CCF-13-49153, CCF-11-17937, CCF-10-17334, and CCF-10-18600.
Tutorial Schedule

2:00  Introduction (5 mins) (RD+SF)
2:05  Introduction to the Boa Language (20 mins) (RD)
2:25  Using Boa in VS Code (10 mins) (RD)
2:35  Hands-on Task 0: Boa+VS Code (20 mins) (SF)
2:55  Overview of the Boa Study Template (15 mins) (SF)
3:10  Hands-on Task 1: A Simple Boa Query (20 mins) (RD+SF)
3:30  Break (30 mins)
4:00  Hands-on Task 2: Adding a Second Query (30 mins) (RD+SF)
4:30  Hands-on Task 3: Analyzing Boa Outputs with Pandas (45 mins)
5:15  Publishing Replication Packages (15 mins) (RD)
5:30  Finished
These Slides

https://go.unl.edu/fse22-slides
Today’s Material

https://cse.unl.edu/~rdyer/boa-fse-tutorial.zip
Tutorial Schedule

2:00  Introduction (5 mins) (RD+SF)
2:05  Introduction to the Boa Language (20 mins) (RD)
2:25  Using Boa in VS Code (10 mins) (RD)
2:35  Hands-on Task 0: Boa+VS Code (20 mins) (SF)
2:55  Overview of the Boa Study Template (15 mins) (SF)
3:10  Hands-on Task 1: A Simple Boa Query (20 mins) (RD+SF)

3:30  Break (30 mins)

4:00  Hands-on Task 2: Adding a Second Query (30 mins) (RD+SF)
4:30  Hands-on Task 3: Analyzing Boa Outputs with Pandas (45 mins)
5:15  Publishing Replication Packages (15 mins) (RD)
5:30  Finished
Boa Website and Credentials

For today, everyone will be given a Boa user/password (handed out).

https://go.unl.edu/boamsr
Install the Boa VS Code Extension

https://go.unl.edu/vscode  https://go.unl.edu/boa-vscode

NOTE: If you are having problems, Sam will be around to try and help.
Tutorial Schedule

2:00  Introduction (5 mins) (RD+SF)
2:05  Introduction to the Boa Language (20 mins) (RD)
2:25  Using Boa in VS Code (10 mins) (RD)
2:35  Hands-on Task 0: Boa+VS Code (20 mins) (SF)
2:55  Overview of the Boa Study Template (15 mins) (SF)
3:10  Hands-on Task 1: A Simple Boa Query (20 mins) (RD+SF)
3:30  Break (30 mins)
4:00  Hands-on Task 2: Adding a Second Query (30 mins) (RD+SF)
4:30  Hands-on Task 3: Analyzing Boa Outputs with Pandas (45 mins)
5:15  Publishing Replication Packages (15 mins) (RD)
5:30  Finished
The Boa Language and Infrastructure
Why mine software repositories?

- Learn from the past
- Keep doing what works
- To find better designs
- Empirical validation
- Spot (anti-)patterns
- Inform the future
Open source repositories

Google code

Apache

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?"
Access repository

mine project metadata

foreach project

Output count of all null checks

Find null checks in each source

mine source code

Find all Java source files

 Fixes bug?

Yes
A solution in Java...

```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, JGit, and Eclipse JDT libraries

Uses Hadoop framework

Explicit/manual parallelization

Too much code! Do not read!
The Boa language and data-intensive infrastructure

https://boa.cs.iastate.edu/
Boa architecture

Boa Language
- MapReduce
- Domain-specific Types
- Visitors

Boa's Compiler
- MapReduce
- Quantifiers
- User Functions
- Visitors
- Cached Data input reader
- Runtime

Boa's Data Infrastructure
- GitHub
- Replicator
- Caching Translator
- Local Cache

1 Pike et al, Scientific Prog. Journal, Vol 13, No 4, 2005
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;
                            }
                    return true;
                }
            }.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, JGit, and Eclipse JDT libraries

Uses Hadoop framework

Explicit/manual parallelization
A better solution...

count: output sum of int;

visit(input, 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 7 lines of code!

Automatically parallelized!

No external libraries needed!

Analyzes 128.6 million source files in about 17 minutes!

(only 8 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[] = 120789791
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 : string
- 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

iskind := function (pat: string, k: dsl_type) : bool {
    return match(format(`^%s`, pat), string(k));
};

Matches a ‘kind’ (like ChangeKind or FileKind) against a string pattern. Can be very useful for things like testing if something is a source file:

iskind("SOURCE_", cf.kind)
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
http://boa.cs.iastate.edu/docs/quantifiers.php

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

\textbf{ifall} (i: \texttt{int}; \texttt{condition}...)  
\hspace{1cm} \texttt{body};

For all values of \textit{i},

if \texttt{condition} holds

then

run \texttt{body} once (with \textit{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

\[
\text{id} := \text{visitor} \{ \\
\quad \text{before } \text{id}:\text{T} \rightarrow \text{statement} \\
\quad \text{after} \quad \text{id}:\text{T} \rightarrow \text{statement} \\
\quad \ldots \\
\}; \\
\text{visit}(\text{startNode}, \text{id}); \\
\]

Execute \text{statement} either \text{before} or \text{after} visiting the children of a node of type \text{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

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

```java
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;
}

Tutorial Schedule

2:00  Introduction (5 mins) (RD+SF)
2:05  Introduction to the Boa Language (20 mins) (RD)
2:25  **Using Boa in VS Code (10 mins) (RD)**
2:35  Hands-on Task 0: Boa+VS Code (20 mins) (SF)
2:55  Overview of the Boa Study Template (15 mins) (SF)
3:10  Hands-on Task 1: A Simple Boa Query (20 mins) (RD+SF)
3:30  Break (30 mins)
4:00  Hands-on Task 2: Adding a Second Query (30 mins) (RD+SF)
4:30  Hands-on Task 3: Analyzing Boa Outputs with Pandas (45 mins)
5:15  Publishing Replication Packages (15 mins) (RD)
5:30  Finished
Install the Boa VS Code Extension

https://go.unl.edu/vscode  https://go.unl.edu/boa-vscode

NOTE: If you are having problems, Sam will be around to try and help.
# support for syntax highlighting

0: output collection of string;

# support for study template substitutions

# support for snippets
# this is an example query

# Counting the 10 most used programming languages

p: Project = input;

counts: output top(10) of string weight int;

foreach (i: int; def(p.programming_languages[i]))
    counts << p.programming_languages[i] weight 1;
# How many AST nodes are in each file in the latest snapshot?

```boa
data output collection[project: string][file: string] of int;
```

```boa
def visit(input, visitor {
  before n: CodeRepository -> {
    snapshot := getSnapshot(n, "SOURCE_");
    foreach (i: int; def(snapshot[i]))
      visit(snapshot[i]);
    stop;
  }

  before cf: ChangedFile -> {
    if (match('ERROR', string(cf.kind)))
      stop;
    astCount = 0;
  }

  after cf: ChangedFile ->
    o[input.id][escape(cf.name)] << astCount;

  # by default, count all visited nodes
  before _ -> astCount++;

  # these nodes are not part of the AST, so do nothing when visiting
  before Person -> ;
```

Ln 28, Col 4 Tab Size: 4 Boa
Tutorial Schedule

2:00  Introduction (5 mins) (RD+SF)
2:05  Introduction to the Boa Language (20 mins) (RD)
2:25  Using Boa in VS Code (10 mins) (RD)
2:35  Hands-on Task 0: Boa+VS Code (20 mins) (SF)
2:55  Overview of the Boa Study Template (15 mins) (SF)
3:10  Hands-on Task 1: A Simple Boa Query (20 mins) (RD+SF)
3:30  Break (30 mins)
4:00  Hands-on Task 2: Adding a Second Query (30 mins) (RD+SF)
4:30  Hands-on Task 3: Analyzing Boa Outputs with Pandas (45 mins)
5:15  Publishing Replication Packages (15 mins) (RD)
5:30  Finished
Putting it all together
(running the motivating example)

http://boa.cs.iastate.edu/
Install the Boa VS Code Extension

https://go.unl.edu/vscode  https://go.unl.edu/boa-vscode

NOTE: If you are having problems, Sam will be around to try and help.
Authentication to Boa

- IDE will ask for user/pw

- Also needed in `.env`:
  ```bash
  BOA_API_USER='USERNAME_HERE'
  BOA_API_PW='PASSWORD_HERE'
  ```

- Can also use system keychain for the password - just add the user to your `.env`
Install Necessary Python Modules

Run:

```bash
pip3 install -r requirements.txt
```
in the tutorial directory.

If you have issues, let us know, we have a Docker-based option available.
Hands-on Task 0

Open the files “boa/queries/task0/part1.boa” and “boa/queries/task0/part2.boa”

We will use these files for the first activity
Recall the task is to answer

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

mine project metadata

foreach project

Output count of all null checks

Find null checks in each source

mine source code

Find all Java source files

Fixes bug?

Yes

mine revisions
Step 1: Declare input and visitor

```java
p: Project = input;

visitor {
}
```
Step 2: Finding null checks

p: Project = input;

visitor {
    # look for expressions of the form:
    #   null == expr OR expr == null
    #   null != expr OR expr != null
}
Step 2: Finding null checks

p: Project = input;

visitor {
    # look for expressions of the form:
    # null == expr OR expr == null
    # null != expr OR expr != null
    before exp: Expression ->

    };

Step 2: Finding null checks

p: Project = input;

visitor {
    # look for expressions of the form:
    # null == expr OR expr == null
    # null != expr OR expr != null
    before exp: Expression ->
        if (exp.kind == ExpressionKind.EQ || exp.kind == ExpressionKind.NEQ)
    }

Step 2: Finding null checks

p: Project = input;

visitor {
    # look for expressions of the form:
    # null == expr OR expr == null
    # null != expr OR expr != null
    before exp: Expression ->
        if (exp.kind == ExpressionKind.EQ || exp.kind == ExpressionKind.NEQ)
            exists (i: int; isliteral(exp.expressions[i], "null"))
};
Step 3: Output null checks count

```java
p: Project = input;

NullChecks: output sum of int;

visitor {
    # look for expressions of the form:
    # null == expr OR expr == null
    # null != expr OR expr != null
    before exp: Expression ->
        if (exp.kind == ExpressionKind.EQ || exp.kind == ExpressionKind.NEQ)
            exists (i: int; isliteral(exp.expressions[i], "null"))
            NullChecks << 1;
};
```
Step 4: Name and call the visitor

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

nullCheckVisitor :=
   visitor {
      # look for expressions of the form:
      #  null == expr OR expr == null
      #  null != expr OR expr != null
      before exp: Expression ->
         if (exp.kind == ExpressionKind.EQ || exp.kind ==
             ExpressionKind.NEQ)
            exists (i: int; isliteral(exp.expressions[i], "null"))
               NullChecks << 1;
   };

visit(p, nullCheckVisitor);
Let’s see it in action!

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

nullCheckVisitor :=
    visitor {
        # look for expressions of the form:
        # null == expr OR expr == null
        # null != expr OR expr != null
        before exp: Expression ->
            if (exp.kind == ExpressionKind.EQ || exp.kind == ExpressionKind.NEQ)
                exists (i: int; isliteral(exp.expressions[i], "null"))
                    NullChecks << 1;
    };

visit(p, nullCheckVisitor);
Access repository

mine project metadata

foreach project

Output count of all null checks

Find null checks in each source

mine source code

Find all Java source files

mine revisions

Fixes bug? Yes

Fixes
Recall the visitor

nullCheckVisitor :=

```java
visitor {
    # look for expressions of the form:
    # null == expr OR expr == null
    # null != expr OR expr != null
    before exp: Expression ->
        if (exp.kind == ExpressionKind.EQ
            || exp.kind == ExpressionKind.NEQ)
            exists (i: int; isliteral(exp.expressions[i], "null"))
                NullChecks << 1;
}
```
nullCheckVisitor := visitor {
    before stmt: Statement ->
        # increase the counter if there is an IF statement
    if (stmt.kind == StatementKind.IF)
        visit(stmt.expression, visitor {
            # where the boolean condition is of the form:
            # null == expr OR expr == null
            # null != expr OR expr != null
            before exp: Expression ->
                if (exp.kind == ExpressionKind.EQ
                    || exp.kind == ExpressionKind.NEQ)
                    exists (i: int; isliteral(exp.expressions[i], "null"))
                    NullChecks <<= 1;
        })
};
Step 6: Make visitor reusable

count := 0;
nullCheckVisitor := visitor {
    before stmt: Statement ->
        # increase the counter if there is an IF statement
        if (stmt.kind == StatementKind.IF)
            visit(stmt.expression, visitor {
                # where the boolean condition is of the form:
                # null == expr OR expr == null
                # null != expr OR expr != null
                before exp: Expression ->
                    if (exp.kind == ExpressionKind.EQ
                        || exp.kind == ExpressionKind.NEQ)
                        exists (i: int; isliteral(exp.expressions[i], "null"))
                        count++;
            });
};
Step 7: Visitor to compare revisions

```java
files: map[string] of ChangedFile;

visit(p, visitor {
    before cf: ChangedFile -> {
        if (haskey(files, cf.name))
            analysis(cf, files[cf.name]); # TODO

        if (cf.change == ChangeKind.DELETED)
            remove(files, cf.name);
        else
            files[cf.name] = cf;
    stop;
    }
})
```
Step 8: Check for bug fixes

```c
files: map[string] of ChangedFile;

visit(p, visitor {
    before cf: ChangedFile -> {
        if (haskey(files, cf.name) && isfixingrevision(current(Revision).log))
            analysis(cf, files[cf.name]); # TODO

        if (cf.change == ChangeKind.DELETED)
            remove(files, cf.name);
        else
            files[cf.name] = cf;

        stop;
    }
});
```
Step 9: Define the analysis

```javascript
analysis := function(cf: ChangedFile, prevCf: ChangedFile) {
    # count how many null checks were previously in the file
    count = 0;
    visit(prevCf, nullCheckVisitor);
    last := count;

    # count how many null checks are currently in the file
    count = 0;
    visit(cf, nullCheckVisitor);

    # if there are more null checks, output
    if (count > last)
        NullChecks << 1;
};
```
This solves the ENTIRE task!

Let’s see it in action!
<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>2:00</td>
<td>Introduction (5 mins) (RD+SF)</td>
</tr>
<tr>
<td>2:05</td>
<td>Introduction to the Boa Language (20 mins) (RD)</td>
</tr>
<tr>
<td>2:25</td>
<td>Using Boa in VS Code (10 mins) (RD)</td>
</tr>
<tr>
<td>2:35</td>
<td>Hands-on Task 0: Boa+VS Code (20 mins) (SF)</td>
</tr>
<tr>
<td><strong>2:55</strong></td>
<td>Overview of the Boa Study Template (15 mins) (SF)</td>
</tr>
<tr>
<td>3:10</td>
<td>Hands-on Task 1: A Simple Boa Query (20 mins) (RD+SF)</td>
</tr>
<tr>
<td>3:30</td>
<td>Break (30 mins)</td>
</tr>
<tr>
<td>4:00</td>
<td>Hands-on Task 2: Adding a Second Query (30 mins) (RD+SF)</td>
</tr>
<tr>
<td>4:30</td>
<td>Hands-on Task 3: Analyzing Boa Outputs with Pandas (45 mins)</td>
</tr>
<tr>
<td>5:15</td>
<td>Publishing Replication Packages (15 mins) (RD)</td>
</tr>
<tr>
<td>5:30</td>
<td>Finished</td>
</tr>
</tbody>
</table>
Code from Task 0 looked different

It has \( \{@\text{findNullExps}\}@ \) instead of a visitor

\[
p: \text{Project} = \text{input};
\]

\[
\text{NullChecks}: \text{output sum of int};
\]

\[
\text{nullCheckVisitor} := \{@\text{findNullExps}\};;
\]

\[
\text{visit}(p, \text{nullCheckVisitor});
\]

That’s a substitution from the study template
Research is more than one question!

And questions can require multiple queries.

A lot of queries to manage!
Queries have common elements

Enter Substitutions!

\[ p: \text{Project} = \text{input}; \]
\[ \text{NullChecks: output sum of int}; \]

\[ \text{nullCheckVisitor} := \{@\text{findNullExps}@\}; \]

\[ \text{visit}(p, \text{nullCheckVisitor}); \]

- Reuse code and avoid copying
- Declared in \texttt{study-config.json}
Boa Output is an unusual format

Conversion to CSV possible, as are other post-processing steps.

Controlled through `study-config.json`
study-config.json?

- Every stage/part of the study
- The “core” of the template

Let’s take a look.
Map short names to full Boa Dataset Names.
Output File

Input Queries
Replace targets
With a file

Or with the contents of a string

Can be for all queries (shown here) or just one (put in query definition)
Run custom processors (like deduplication) on a query output

Generate CSV (handle Boa output format)
Analysis script name (in analyses dir)

Depend on necessary inputs: can just run analysis, inputs will be built if needed
Tutorial Schedule

2:00  Introduction (5 mins) (RD+SF)
2:05  Introduction to the Boa Language (20 mins) (RD)
2:25  Using Boa in VS Code (10 mins) (RD)
2:35  Hands-on Task 0: Boa+VS Code (20 mins) (SF)
2:55  Overview of the Boa Study Template (15 mins) (SF)
3:10  Hands-on Task 1: A Simple Boa Query (20 mins) (RD+SF)
3:30  Break (30 mins)
4:00  Hands-on Task 2: Adding a Second Query (30 mins) (RD+SF)
4:30  Hands-on Task 3: Analyzing Boa Outputs with Pandas (45 mins)
5:15  Publishing Replication Packages (15 mins) (RD)
5:30  Finished
Hands-on Task 1

Open the file “boa/queries/task1/part1.boa”

This query looks at commit timestamps and finds ones that are suspiciously old (older than the date CVS was released).
Looking for Suspicious Commits

```python
# output a list of timestamps found
P: output collection[projUrl: string][revId: string] of time;
    # indexed by project URL and commit hash
visit(input, visitor {
    # look at each revision node
    before r: Revision ->
        # is the commit date old?
        if (r.commit_date < T"November 19, 1990, 00:00:00 AM UTC")
            # output what we found
            P[input.project_url][r.id] << r.commit_date;
});
```

DOWNLOAD THE TASK1/PART1.TXT OUTPUT
Why is there no output?

Answer: Because the small dataset has no commits with suspiciously old timestamps!

Let’s modify the `study-config.json` file and add a second dataset:

“2019 October/GitHub (medium)”

and then re-run the analysis using that new dataset.
Edit study-config.json

1) Add a new dataset. Datasets can have any name (the key), but must have specific values. The IDE should autocomplete values for you.

2) Modify the output file for Task 1 so it uses the new dataset you specified.
Now that it runs, analyze it!

Look for the generated outputs in tables/ and figures/
Hands-on Task 1, Part 2

Make a copy of the `task1/part1.boa` file.

Be sure to add the new query to your `study-config.json`!

Now edit the query so it only looks at commits containing `at least one source file`.
Hands-on Task 1, Part 2: Hints

1. Hint: is there a domain-specific function to determine if a file is a specific kind?

2. Hint: can you quantify if such a file exists in a Revision?
Hands-on Task 1, Part 2: Solution

P: output collection[projUrl: string][revId: string] of time;

visit(input, visitor {
  before r: Revision ->
    exists (i: int; iskind("SOURCE_", r.files[i].kind))
      if (r.commit_date < T"November 19, 1990, 00:00:00 AM UTC")
        P[input.project_url][r.id] << r.commit_date;
});
Task 1 Solution

https://go.unl.edu/fse-task1
Tutorial Schedule

2:00 Introduction (5 mins) (RD+SF)
2:05 Introduction to the Boa Language (20 mins) (RD)
2:25 Using Boa in VS Code (10 mins) (RD)
2:35 Hands-on Task 0: Boa+VS Code (20 mins) (SF)
2:55 Overview of the Boa Study Template (15 mins) (SF)
3:10 Hands-on Task 1: A Simple Boa Query (20 mins) (RD+SF)

3:30 Break (30 mins)

4:00 Hands-on Task 2: Adding a Second Query (30 mins) (RD+SF)
4:30 Hands-on Task 3: Analyzing Boa Outputs with Pandas (45 mins)
5:15 Publishing Replication Packages (15 mins) (RD)
5:30 Finished
Tutorial Schedule

2:00  Introduction (5 mins) (RD+SF)
2:05  Introduction to the Boa Language (20 mins) (RD)
2:25  Using Boa in VS Code (10 mins) (RD)
2:35  Hands-on Task 0: Boa+VS Code (20 mins) (SF)
2:55  Overview of the Boa Study Template (15 mins) (SF)
3:10  Hands-on Task 1: A Simple Boa Query (20 mins) (RD+SF)
3:30  Break (30 mins)

4:00  Hands-on Task 2: Adding a Second Query (30 mins) (RD+SF)
4:30  Hands-on Task 3: Analyzing Boa Outputs with Pandas (45 mins)
5:15  Publishing Replication Packages (15 mins) (RD)
5:30  Finished
Hands-on Task 2

Open the file “boa/queries/task2/task2.boa”

Open the file “analyses/task2.py”

The goal of this research question is to see how often Java projects declare enum types. Here we provide the analysis, but you need to write the query.
Hands-on Task 2: Expected Output

\begin{tabular}{lr}
  \textbf{enums} & \textbf{count} \\
  & 78.00 \\
  \textbf{mean} & 20.64 \\
  \textbf{std}  & 93.07 \\
  \textbf{min}  & 1.00 \\
  \textbf{25\%} & 1.00 \\
  \textbf{50\%} & 2.00 \\
  \textbf{75\%} & 6.75 \\
  \textbf{max}  & 766.00 \\
\end{tabular}
Hands-on Task 2: Expected Output

![Graph showing expected output](image)
Hands-on Task 2: Expected Output

![Histogram](image)
Hands-on Task 2: TODO 1 Hints

1. Hint: look at the analysis - how does it expect the CSV data to be organized?

2. Hint: define your output variable to match!
# expected output has 4 columns:

names=['var', 'project', 'file', 'enums']

# first is just the output variable’s name
# second is the project (id or name – anything unique)
# third is the file name
# fourth is the number of enums declared in that file

# the last column is always the aggregated value
# so we need 2 indexes:

o: output sum[proj: string][file: string] of int;
Hands-on Task 2: TODO 2 Hints

1. Hint: can you get just the last snapshot for each project?

2. Hint: enums are **Declarations**, can you find them?

3. Hint: every time you find an enum, output
Hands-on Task 2: TODO 2 Solution

```javascript
visit(input, visitor {
  
});
```
Hands-on Task 2: TODO 2 Solution

```go
visit(input, visitor {  
    before node: CodeRepository -> {  
        snapshot := getsnapshot(node);  
        foreach (i: int; def(snapshot[i]))  
            visit(snapshot[i]);  
        stop;
    }
});
```
Hands-on Task 2: TODO 2 Solution

```java
visit(input, visitor { 
    before node: CodeRepository -> {
        snapshot := getsnapshot(node);
        foreach (i: int; def(snapshot[i]))
            visit(snapshot[i]);
        stop;
    }
    before n: Declaration ->
        if (n.kind == TypeKind.ENUM)
            o[input.id][current(ChangedFile).name] << 1;
});
```
Hands-on Task 2: Analysis won’t run

It needs `dupes.csv`, which isn’t generated anywhere.

Hint: Use a processor and the `hashes.boa` query.
Hands-on Task 2: Analysis Fix

Add this to queries in study-config.json

```
"hashes.txt": {
    "query": "queries/hashes.boa",
    "dataset": "java",
    "processors": {
        "gendupes.py": {
            "output": "data/txt/dupes.txt",
            "csv": "dupes.csv",
            "cacheclean": [
                "*-deduped.parquet"
            ]
        }
    }
}
```

https://go.unl.edu/dedupe
Task 2: Solution

https://go.unl.edu/fse-task2
Tutorial Schedule

2:00  Introduction (5 mins) (RD+SF)
2:05  Introduction to the Boa Language (20 mins) (RD)
2:25  Using Boa in VS Code (10 mins) (RD)
2:35  Hands-on Task 0: Boa+VS Code (20 mins) (SF)
2:55  Overview of the Boa Study Template (15 mins) (SF)
3:10  Hands-on Task 1: A Simple Boa Query (20 mins) (RD+SF)
3:30  Break (30 mins)
4:00  Hands-on Task 2: Adding a Second Query (30 mins) (RD+SF)
4:30  Hands-on Task 3: Analyzing Boa Outputs with Pandas (45 mins)
5:15  Publishing Replication Packages (15 mins) (RD)
5:30  Finished
Hands-on Task 3

Open the file “boa/queries/task3/counts.boa”

This query counts some things in the dataset we are analyzing, like the number of files and projects.
Hands-on Task 3

Open the file “boa/queries/task3/task3.boa”

This query looks at Java features and finds code that could utilize newer language features.
Hands-on Task 3

In this task, we give you a Boa query from an ICSE 2014 paper.

Your task is to provide the Python analysis that takes the query output and generates a LaTeX table similar to the following:

```
\begin{tabular}{|c|c|c|c|c|c|c|c|}
\hline
 & Assert & Varargs & Binary Literals & Diamond & MultiCatch & Try & with Resources & Underscore Literals \\
\hline
Old & 89K & 612K & 56K & 3.3M & 341K & 489K & 22.2M \\
New & 291K & 1.6M & 5K & 414K & 24K & 33K & 2.3M \\
\hline
All Files & 380K & 2.2M & 61K & 3.7M & 365K & 522K & 24.5M \\
Projects & 18.18\% & 12.74\% & 0.11\% & 12.25\% & 2.28\% & 1.85\% & 20.17\% \\
\hline
\end{tabular}
```

Figure 15: Potential language feature uses, in old files (before feature release) and new files (after feature release).
Hands-on Task 3

Open the file “analyses/task3.py”

This is the skeleton of the analysis for Task 3.

Read the TODO items and attempt to write Python code for each.

You may want to look at other analyses we provided for insights.
Hands-on Task 3: TODO 1 Hints

load `counts.csv` into a Pandas DataFrame

1. Hint: maybe you can drop a column?

2. Hint: you can use `set_index()` to make one of the columns into the dataframe’s index.

3. Hint: you can use `df[col][row]` to select a cell.
Hands-on Task 3: TODO 1 Solution

defcounts = get_df('counts',
    'task3',
    drop=['idx'],
    names=['var', 'idx', 'count'])

defcounts = defcounts.set_index('var')

# get the total number of studied projects
total_projs = defcounts['count']['StudiedProjects']

# get the total number of Java files
total_files = defcounts['count']['JavaFiles']
Hands-on Task 3: TODO 2 Hints

1. load `task3.csv` into a Pandas DataFrame

2. Hint: do we need to drop a column?
Hands-on Task 3: TODO 2 Solution

def = get_df(filename='task3', # reads task3.csv
    subdir='task3', # from the task3 sub-directory
    # so the file is: data/csv/task3/task3.csv
    
    names=['var', 'kind', 'count']
)

Handson Task 3: TODO 3 Hints

1. Hint: create a style for the table

2. Hint: you can use `save_table()` to generate a LaTeX table in a file
Hands-on Task 3: TODO 3 Solution

```python
style = highlight_rows(highlight_cols(get_styler(df4)))

save_table(style,
    'task3.tex',
    colsep='1pt',
    rotate=50,
    rowcolors=True,
    column_format='|c|r|r|r|r|r|r|r|
)
```
Hands-on Task 3: Add to study-config.json

Add this analyses in study-config.json

```json
"task3.py": {
  "input": [
    "task3/task3.csv",
    "task3/counts.csv"
  ]
}
```

https://go.unl.edu/task3-analysis
Task 3: Solution

https://go.unl.edu/fse-task3
Tutorial Schedule

2:00  Introduction (5 mins) (RD+SF)
2:05  Introduction to the Boa Language (20 mins) (RD)
2:25  Using Boa in VS Code (10 mins) (RD)
2:35  Hands-on Task 0: Boa+VS Code (20 mins) (SF)
2:55  Overview of the Boa Study Template (15 mins) (SF)
3:10  Hands-on Task 1: A Simple Boa Query (20 mins) (RD+SF)
3:30  Break (30 mins)
4:00  Hands-on Task 2: Adding a Second Query (30 mins) (RD+SF)
4:30  Hands-on Task 3: Analyzing Boa Outputs with Pandas (45 mins)
5:15  Publishing Replication Packages (15 mins) (RD)
5:30  Finished
Open Science Is Important
Your research. Preserved.

Upload
- Any size/format
- Any science
- Any research output
- Dropbox

Describe
- Reusable for others
- Link to related research
- Open, embargoed and private content

Publish
- Instantly available
- DOI: Citable. Discoverable.
- Article Level Metrics

zenodo
Research. Shared.

AN OPEN DEPENDABLE HOME FOR THE LONGTAIL OF SCIENCE.

Enabling sharing and preservation of multidisciplinary research results.
1) Go to: https://sandbox.zenodo.org/

2) Make a user (you can’t use OAuth logins - make an actual user!)

3) Go to your user’s “Applications” settings
4) Create a “New token”

You have not yet created any personal access tokens. Click the 'New token' button to create a personal access token.
5) Give it a name
6) Select first two scopes

Copy the token for later - you can’t access it again!
Environment Setup

Edit the “.env” file to put your token:

ZENODO_API_TOKEN='<paste here>'
ZENODO_API_ENDPOINT='https://sandbox.zenodo.org'
Edit the JSON

Edit the "zenodo.json" file.

Fill in/change anything you see to match your particular paper.
Upload to Zenodo

Now you can upload to Zenodo!

Run:  make package and then make zenodo

This will generate the ZIP files, create a new deposit, and upload the ZIPs. If you run it again, it will update the existing record’s metadata and re-upload any files that changed.

The record is not published yet!
After Publication

Once your paper is accepted, you need to un-blind your record.

1) Re-edit the .zenodo.json file to update the author information
2) Re-run “make zenodo”
3) Update your paper to reference the DOI (if you have not already)
Boa

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