Model-based Mining of Software Repositories

Markus Scheidgen
Agenda

- Mining Software Repositories (MSR) and current approaches
- *srcrepo* – a model-based MSR system
  - *srcrepo* components and analysis process
  - a meta-model for *source code repositories*
  - gathering software metrics with an *OCL-like* internal Scala DSL
- **work in progress** - discussion of remaining problems and limitations
Relevant Research Fields

**Mining Software Repositories (MSR)**
The term *mining software repositories* (MSR) has been coined to describe a broad class of investigations into the examination of *software repositories*. The premise of MSR is that *empirical and systematic investigations* of repositories will shed new light on the process of *software evolution*. [1]

**Software Metrics**
A software metric is a mathematical definition mapping the entities of a software system to numeric metrics values. [...] to express features of software with numbers in order to facilitate *software quality assessment*. [2]

**Reverse Engineering**
*Reverse engineering* is the process of analyzing a subject system to (1) identify the system's components and their interrelationships and (2) create representations of the system in another form or at a higher level of abstraction [3]

**Software Evolution Research (SER)**
- (dis-)proving *Lehmann’s Laws* of software evolution
- empirical investigations of software repositories through statistical analysis of software and software change metrics over the evolutionary cause of many software systems.

**Model-based Mining Software Repositories (with srcrepo)**
Overcoming *heterogeneity* and *accessibility* by raising the level of abstraction, while ensuring *scalability* and retaining meaningful *information depth*.

**Contemporary Approaches to Large Scale MSR for SER**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>■ database for over 3000 open source software projects</td>
<td>■ database and searchable index of declarations from over 4000 Java software projects</td>
<td>■ domain specific language (DSL) for mining meta-data in ultra-large software repositories</td>
</tr>
<tr>
<td>■ contains data about all revisions</td>
<td>■ tracks only release revisions</td>
<td>■ only tracks VCS meta-data, e.g. “How many revisions are there in all Java projects using SVN?”</td>
</tr>
<tr>
<td>■ <em>Alitheia</em>, multiple <em>version control systems</em> (VCS), but only text-based metrics</td>
<td>■ metrics based on declarations (classes, methods, fields, etc., e.g. CK-metrics), but not based on actual implementations (e.g. McCabe, Halstead)</td>
<td></td>
</tr>
<tr>
<td>■ not only <em>source code repositories</em> (SCR) via VCS, also issue-tracking systems, mailing-lists, etc.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scalability</th>
<th>Heterogeneity</th>
<th>Accessibility</th>
<th>Information Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>■ a project</td>
<td>■ abstraction from VCS [1,2,3]</td>
<td>■ programming</td>
<td>■ all revisions [1,3], sample revisions [2]</td>
</tr>
<tr>
<td>■ ultra-large scale: 100k+ unrelated projects with varying quality [1,2,3]</td>
<td></td>
<td>■ DSL [3]</td>
<td>■ text [1]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>■ declarations [2]</td>
</tr>
</tbody>
</table>

1. **G. Gousios, D. Spinellis**: *Alitheia core: An extensible software quality monitoring platform*; Proceedings of the 31st International Conference on Software Engineering; 2009
# Goals and Hypothesis

<table>
<thead>
<tr>
<th>Scalability</th>
<th>Heterogeneity</th>
<th>Accessibility</th>
<th>Information Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>■ a project</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>■ large scale: multiple related projects, e.g. Apache, Eclipse</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>■ ultra-large scale: 100k+ unrelated projects with varying quality [1,2,3]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>■ abstraction from VCS [1,2,3]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>■ abstraction from programming language: only metadata [3] or text [1]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>■ programming</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>■ database with index [1,2]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>■ DSL [3]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>■ all revisions [1,3], sample revisions [2]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>■ meta-data [3]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>■ text [1]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>■ declarations [2]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Goals and Hypothesis

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Scalability</th>
<th>Heterogeneity</th>
<th>Accessibility</th>
<th>Information Depth</th>
</tr>
</thead>
</table>
| **Goals** | ■ a project  
■ large scale: multiple related projects, e.g. Apache, Eclipse  
■ ultra-large scale: 100k+ unrelated projects with varying quality [1,2,3] | ■ abstraction from VCS [1,2,3]  
■ abstraction from programming language: only metadata [3] or text [1] | ■ programming  
■ database with index [1,2]  
■ DSL [3] | ■ all revisions [1,3], sample revisions [2]  
■ meta-data [3]  
■ text [1]  
■ declarations [2] |
| **Hypothesis** | ■ cluster- (batching) and cloud- (Map/Reduce)-computing support  
■ distributable databases | ■ common meta-model for VCSs  
■ meta-models for programming languages  
■ common meta-model for metrics | ■ internal DSL: DSL + programming with models  
■ common modeling framework  
■ existing tools/frameworks | ■ all revisions  
■ *abstract syntax trees* (AST)  
■ differences between revisions (e.g. metrics on adaptations and refactorings) |
| **Approaches** | ■ distributable model persistence  
■ distributed processing of models | ■ abstraction for different VCSs exists  
■ abstraction regarding metrics for diff. progr. languages exists  
■ abstraction for diff. languages exists | ■ is there a reasonable programming abstraction for gathering metrics/change metrics |
srcrepo – Components and Process
srcrepo – Components and Process

- Large scale software repositories (e.g. GitHub, SourceForge)
- Software projects
  - Source code repository (e.g. controlled by Git, SVN, CVS)
  - Source code (e.g. Java, C++, Eclipse*)
  - Issue tracker, mailing lists, wiki
srcrepo – Components and Process

large scale software repositories
(e.g. github, sourceforge)

software projects

source code repository
(e.g. controlled by Git, SVN, CVS)

source code
(e.g. java, C++, eclipse*)

issue tracker, mailing lists, wiki

srcrepo storage
(EMF-models via EMF-Fragments, e.g. on mongodb)

revision tree

import

AST-models of new and changed CUs

Saturday, 27. September 2014
srcrepo – Components and Process

large scale software repositories (e.g. github, sourceforge)

software projects

source code repository (e.g. controlled by Git, SVN, CVS)

source code (e.g. java, C++, eclipse*)

issue tracker, mailing lists, wiki

srcrepo storage (EMF-models via EMF-Fragments, e.g. on mongodb)

revision tree

import

AST-models of new and changed CUs

srcrepo runtime (headless eclipse RCP)

revision tree

analysis

fully resolved snapshot models
**srcrepo – Components and Process**

**large scale software repositories**
(e.g. github, sourceforge)

**srcrepo storage**
(EMF-models via EMF-Fragments, e.g. on mongodb)

**srcrepo runtime**
(headless eclipse RCP)

Software projects

Source code repository
(e.g. controlled by Git, SVN, CVS)

Source code
(e.g. java, C++, eclipse*)

Issue tracker, mailing lists, wiki

Revision tree

Revision tree

AST-models of new and changed CUs

Analysis

Fully resolved snapshot models

Import
large scale software repositories (e.g. github, sourceforge)

source code (e.g. java, C++, eclipse*)

issue tracker, mailing lists, wiki

source code repository (e.g. controlled by Git, SVN, CVS)

import

AST-models of new and changed CUs

analysis

fully resolved snapshot models

revision tree

source code

revision tree

SAT-models of new and changed CUs

revision tree
large scale software repositories (e.g. github, sourceforge)
srcrepo storage (EMF-models via EMF-Fragments, e.g. on mongodb)
srcrepo runtime (headless eclipse RCP)
revisions
sources
metrics

issue tracker, mailing lists, with

source code (e.g. java, C++, eclipse*)

source code (e.g. controlled by Git, SVN, CVS)

import

AST-models of new and changed CUs

analysis

fully resolved snapshot models

1 2 3

OCL

1 2 3

M1 M2 M3

1 2 3

S1 S2 S2

References:

AST-models of new and changed CUs

1 2 3

OCL

1 2 3

M1 M2 M3

Saturday, 27. September 2014
issue tracker, mailing lists, wiki
(source code (e.g. java, C++, eclipse*))

large scale software repositories (e.g. github, sourceforge)

srcrepo storage (EMF-models via EMF-Fragments, e.g. on mongodb)

srcrepo runtime (headless eclipse RCP)

revisions

sources

metrics

AST-models of new and changed CUs

import

analysis

revision tree

fully resolved snapshot models

source code repository (e.g. controlled by Git, SVN, CVS)

source code (e.g. java, C++, eclipse*)
Large scale software repositories (e.g. GitHub, SourceForge) store source code (e.g. Java, C++, Eclipse*), issue tracker, mailing lists, and wiki. Source code is controlled by Git, SVN, or CVS. Software projects have an import stage for AST-models of new and changed change units (CUs). An analysis stage is followed by fully resolved snapshot models. Metrics and timelines are stored and compared using EMF-Compare and OCL.
source code
(e.g. java, C++, eclipse*)

issue tracker, mailing lists, with
(source code
(e.g. controlled by Git, SVN, CVS))

statistics software
(e.g. R, Matlab)

source code
repository
(e.g. controlled by Git, SVN, CVS)

large scale software
repositories
(e.g. github, sourceforge)

srcrepo
storage
(EMF-models via EMF-Fragments,
e.g. on mongodb)

srcrepo
runtime
(headless eclipse RCP)

revisions

issues
tracker, mailing lists, wiki

software projects

revision tree

AST-models of new and changed CUs

import

analysis

EMF-Compare

OCL

store

timelines of metrics

export

fully resolved snapshot
models

One superior

Two superior

Three superior

Saturday, 27. September 2014
“Demo”
A OCL-like internal Scala DSL for Computing Metrics

- **OCL-like** internal Scala DSL analog to our internal Scala model transformation language [1]
- OCL collection operations mapped to Scala’s higher-order functions [2]:

1. **L. George, A. Wider, M. Scheidgen**: Type-Safe Model Transformation Languages as Internal DSLs in Scala; Theory and Practice of Model Transformations - 5th International Conference, ICMT; 2012
2. **Filip Krikava**: Enrichting EMF Models with Scala; Slideshare
A OCL-like internal Scala DSL for Computing Metrics

- **OCL-like internal Scala DSL analog to our internal Scala model transformation language [1]**

- **OCL collection operations mapped to Scala’s higher-order functions [2]:**

  ```
  context Model:
  self.ownedElements->collect(plp.ownedElements)->size
  ```

1. **L. George, A. Wider, M. Scheidgen:** *Type-Safe Model Transformation Languages as Internal DSLs in Scala*; Theory and Practice of Model Transformations - 5th International Conference, ICMT; 2012

2. **Filip Krikava:** *Enriching EMF Models with Scala*; Slideshare
A OCL-like internal Scala DSL for Computing Metrics

- **OCL-like** internal Scala DSL analog to our internal Scala model transformations [1]

- **OCL collection operations** mapped to Scala’s higher-order functions [2]:

  ```scala
  context Model:
  self.ownedElements->collect(p|p.ownedElements)->size
  
  def numberOfFirstPackageLevelTypes(self: Model): Int =
  self.getOwnedElements().collect(p=>p.getOwnedElements()).size()
  ```

1. **L. George, A. Wider, M. Scheidgen**: *Type-Safe Model Transformation Languages as Internal DSLs in Scala*; Theory and Practice of Model Transformations - 5th International Conference, ICMT; 2012
2. **Filip Krikava**: *Enriching EMF Models with Scala*; Slideshare

Saturday, 27. September 2014
A OCL-like internal Scala DSL for Computing Metrics

- Extending OCL’s collection operations:
  - convenience operations
  - closure
  - aggregation
  - execution

```scala
trait OclCollection[E] extends java.lang.Iterable[E]
{
  def size(): Int
  def first(): E
  def exists(predicate: (E) => Boolean): Boolean
  def forAll(predicate: (E) => Boolean): Boolean
  def select(predicate: (E) => Boolean): OclCollection[E]
  def reject(predicate: (E) => Boolean): OclCollection[E]
  def collect[R](expr: (E) => R): OclCollection[R]
  def selectOfType[T]:OclCollection[T]
  def collectNotNull[R](expr: (E) => R): OclCollection[R]
  def collectAll[R](expr: (E) => OclCollection[R]): OclCollection[R]
  def closure(expr: (E) => OclCollection[E]): OclCollection[E]
}
```
    def size(): Int
    def first(): E
    def exists(predicate: (E) => Boolean): Boolean
    def forAll(predicate: (E) => Boolean): Boolean
    def select(predicate: (E) => Boolean): OclCollection[E]
    def reject(predicate: (E) => Boolean): OclCollection[E]
    def collect[R](expr: (E) => R): OclCollection[R]
    def selectOfType[T]: OclCollection[T]
    def collectNotNull[R](expr: (E) => R): OclCollection[R]
    def collectAll[R](expr: (E) => OclCollection[R]): OclCollection[R]
    def closure(expr: (E) => OclCollection[E]): OclCollection[E]
    def sum(expr: (E) => Double): Double
    def product(expr: (E) => Double): Double
    def max(expr: (E) => Double): Double
    def min(expr: (E) => Double): Double
    def stats(expr: (E) => Double): Stats
    def run(runnable: (E) => Unit): Unit
}
WMC is the first CK-metric [1]. There different commonly used weights; here we use cyclomatic complexity.

```python
def classes(model: Model): OclCollection[ClassDeclaration] =
    model.getOwnedElements()
    .collectClosure(pkg=>pkg.getOwnedPackages())
    .collectAll(pkg=>pkg.getOwnedElements())
    .collectClosure(typeDcl=>
        typeDcl.getBodyDeclarations()
        .selectOfType[ClassDeclaration])

def WMC(model: Model): Double =
    classes(model).stats(clazz=>
        clazz.getBodyDeclarations()
        .selectOfType[MethodDeclaration]().sum(method=>cyclomaticComplexity(method))).average

def cyclomaticComplexity(method: MethodDeclaration): Int =
    ...
```

Complex Example: Average Weighted Methods per Class (WMC)

WMC is the first CK-metric [1]. There different commonly used weights; here we use cyclomatic complexity.

```python
def classes(model: Model): OclCollection[ClassDeclaration] =
    model.getOwnedElements()  
    .collectClosure(pkg => pkg.getOwnedPackages())  
    .collectAll(pkg => pkg.getOwnedElements())  
    .collectClosure(typeDcl =>
        typeDcl.getBodyDeclarations()  
        .selectOfType[ClassDeclaration])

def WMC(model: Model): Double =
    classes(model).stats(clazz =>
        clazz.getBodyDeclarations()  
        .selectOfType[MethodDeclaration]()  
        .sum(method =>
            cyclomaticComplexity(method)))  
        .average

def cyclomaticComplexity(method: MethodDeclaration): Int =
    ...
```

Complex Example: Average Weighted Methods per Class (WMC)

- WMC is the first CK-metric [1]. There are different commonly used weights; here we use cyclomatic complexity.

```python
def classes(model: Model): OclCollection[ClassDeclaration] =
    model.getOwnedElements().collectClosure(pkg =>
        pkg.getOwnedPackages().collectAll(pkg =>
            pkg.getOwnedElements().collectClosure(typeDcl =>
                typeDcl.getBodyDeclarations().selectOfType[ClassDeclaration]()))
    .stats(clazz =>
        clazz.getBodyDeclarations().selectOfType[MethodDeclaration]().sum(method =>
            cyclomaticComplexity(method))).average

def WMC(model: Model): Double =
classes(model).stats(clazz =>
    clazz.getBodyDeclarations().selectOfType[MethodDeclaration]().sum(method =>
        cyclomaticComplexity(method))).average

def cyclomaticComplexity(method: MethodDeclaration): Int =
   ...
```

**Implementation of the OCL-Collection Operations**

- *Just in time* iterator-based implementation rather than straightforward aggregation of result collections.
### Future Work, Remaining Problems, and Limitations

<table>
<thead>
<tr>
<th>Scalability</th>
<th>Heterogeneity</th>
<th>Accessibility</th>
<th>Information Depth</th>
</tr>
</thead>
</table>
| - very large compilation units  
- incremental snapshot creation  
- batching OCL execution  
- experiments with large scale repository (e.g. *git.eclipse.org*) | - MoDisco for different programming languages  
- common metrics meta-model (e.g. OMG, KDM)  
- VCS abstraction and support for different VCS | - relating results to software repository entities  
- persisting and exporting results | - *diff-models* from comparison of compilation units |

- **Very large compilation units (CU):** e.g. a 3 MB, 600 kLOC CU in *org.eclipse.emf*
  - tends to have lots of dependencies → changes often → makes problem even bigger
  - CUs are smallest common denominator between text-based VCS view and syntax-based AST view
  - smaller units require model-comparison or text-to-AST mappings

- **Support for different programming languages:** either abstraction, parallel meta-models, or mixed approach
  - MoDisco is *extendable*, but only Java support exists; other languages need to be implemented → parallel meta-models
  - A reasonable abstraction for multiple (or all) programming language probably does not exist.
  - A shared abstract meta-model that all language meta-models extends could be an sensible compromise.
Overall model-based MSR with *srcrepo* works, but it still needs work.

80/20: Uncommonly large CUs are problematic and require complex additions to *srcrepo*. Ignored for now.

Main goal *heterogeneity* is theoretically plausible, but requires lots of efforts to show practically. Not a matter of *if*, but of *how much*.

Large experiments are still unfeasible due to lots of small issues rooted in the engineering complexity of the subject matter.