Provenance Support for Rework

Xiang Zhao\textsuperscript{1}, Leon J. Osterweil\textsuperscript{1}, Barbara Staudt Lerner\textsuperscript{2}, Emery R. Boose\textsuperscript{3}, Aaron M. Ellison\textsuperscript{3}

\textsuperscript{1}University of Massachusetts Amherst  
\textsuperscript{2}Mount Holyoke College  
\textsuperscript{3}Harvard University

4th USENIX Workshop on the Theory and Practice of Provenance (TaPP ’12)  
Boston, MA

June 16, 2012
The Problem: How to Support Rework

- Rework is quite common in software development processes
  - Inconsistencies between requirement and design specifications cause reconsideration of both
  - Inconsistencies between code and design too
  - Most software engineering books ignore the topic
- Process provenance support could help
  - People could review earlier decisions to facilitate rework
Refactoring as an Example of Rework

- Refactoring is rework of design
  - May or may not be triggered when code is recognized as being untidy
  - There are many different design patterns [Fowler 1999]
Refactoring is rework of design
- May or may not be triggered when code is recognized as being untidy
- There are many different design patterns [Fowler 1999]

Separate Query from Modifier Refactoring
- Splits a method that was both a query and a modifier into two methods
  - Create a query method to return the same value
  - Change the return statement in original method to return the query
  - Add calls to the query before the calls to the original method
  - Change the original method to void and remove its return statements
Refactoring as an Example of Rework

- Refactoring is rework of design
  - May or may not be triggered when code is recognized as being untidy
  - There are many different design patterns [Fowler 1999]

- Separate Query from Modifier Refactoring
  - Splits a method that was both a query and a modifier into two methods
    - Create a query method to return the same value
    - Change the return statement in original method to return the query
    - Add calls to the query before the calls to the original method
    - Change the original method to void and remove its return statements

- Executing this rework process can entail carrying out a number of different kinds of rework
Exception instances are handled differently according to their types.

Each exception instance triggers rework.
Exception instances are handled differently according to their types.

Each exception instance triggers rework.
Rework is modeled very accurately as recursive step invocations. Actual rework should be guided by context provided by provenance.

Xiang Zhao¹, Leon J. Osterweil¹, Barbara Staudt Lerner², Emery R. Boose³, Aaron M. Ellison³

Provenance Support for Rework
Rework is modeled very accurately as recursive step invocations.
Rework is modeled very accurately as recursive step invocations.

Actual rework should be guided by context provided by provenance.
Typical Questions that Users Want Answers to during Rework

- Where am I?
- What am I doing here?
- How did I get here?
- What have I already tried?
- How did that work out?
- What alternatives do I have now?
- Which are likely to turn out best?
Typical Questions that Users Want Answers to during Rework

- Where am I?
- What am I doing here?
- How did I get here?
- What have I already tried?
- How did that work out?
- What alternatives do I have now?
- Which are likely to turn out best?

Contextual information provided by provenance could help
Context Information that could Help

- Present process execution state
  - Current artifact values
  - Pointers to executing steps and their recursions
- A complete process execution history
  - Prior values of artifacts
  - Previous step execution sequences
- Information that could help to form a plan for completing rework successfully

**Data Derivation Graph** is the key artifact
Defined templates for translating Little-JIL step executions into DAG fragments
Data Derivation Graph

- Defined templates for translating Little-JIL step executions into DAG fragments
- Basic Features
  - Represents how artifacts are derived from each other
  - Incorporates scoping, nesting, hierarchy information

Additional Features
- Links to previous artifacts values
- Detailed history is inferable
- Can generate DDGs dynamically while the process is executing

Defined templates for translating Little-JIL step executions into DAG fragments

Basic Features
- Represents how artifacts are derived from each other
- Incorporates scoping, nesting, hierarchy information

Defined templates for translating Little-JIL step executions into DAG fragments

Basic Features
- Represents how artifacts are derived from each other
- Incorporates scoping, nesting, hierarchy information
Defined templates for translating Little-JIL step executions into DAG fragments

Basic Features
- Represents how artifacts are derived from each other
- Incorporates scoping, nesting, hierarchy information

Additional Features
- Links to previous artifacts values
- Detailed history is inferable
Defined templates for translating Little-JIL step executions into DAG fragments

Basic Features
- Represents how artifacts are derived from each other
- Incorporates scoping, nesting, hierarchy information

Additional Features
- Links to previous artifacts values
- Detailed history is inferable

Can generate DDGs dynamically while the process is executing
Q: What did I do to (the same part of) the source code when I was trying to fix an issue caused by test case failure, which may possibly be the reason why the compilation fails right now?
Q: What did I do to (the same part of) the source code when I was trying to fix an issue caused by test case failure, which may possibly be the reason why the compilation fails right now?

A: See the DDG
Q: What did I do to (the same part of) the source code when I was trying to fix an issue caused by test case failure, which may possibly be the reason why the compilation fails right now?
Scientific Data Process

Check our paper and poster for details

Xiang Zhao¹, Leon J. Osterweil¹, Barbara Staudt Lerner², Emery R. Boose³, Aaron M. Ellison³

Provenance Support for Rework
Future Work

- How to present the “right” information?
  - How to support asking questions during rework?
  - How to make sure the answers are presented in a meaningful way?

Ripple effects

Support for helping users decide the order in which to handle exceptions when many are thrown

Probably can use prospection for this

Study more refactoring patterns

Xiang Zhao\textsuperscript{1}, Leon J. Osterweil\textsuperscript{1}, Barbara Staudt Lerner\textsuperscript{2}, Emery R. Boose\textsuperscript{3}, Aaron M. Ellison\textsuperscript{3}
Future Work

- How to present the “right” information?
  - How to support asking questions during rework?
  - How to make sure the answers are presented in a meaningful way?

- Ripple effects
  - Support for helping users decide the order in which to handle exceptions when many are thrown
    - Probably can use prospection for this

- Study more refactoring patterns
Rework Formalization
- [Cass et al. EWSPT] proposed initial approaches of formalizing rework
- A pattern for modeling rework [Cass et al. ICSP '09]

Context Support
- [Antunes et al. AITSE '10] proposed a context model in software development with multiple layers and perspectives.
- Mylyn [12] is a tool integrating task management and task context [Kersten et al. AOSD '05]

Workflow Provenance: VisTrails [Callahan et al. SIGMOD '06], Kepler [Altintas et al. SSDBM '04], and etc.
Conclusion

- Executable model of rework processes
- Provenance as a first class data
  - Available process wide
  - Directly supports the process where it comes from
- DDG facilitates provenance support
  - Scoping and nesting
  - Version edges and equivalence edges
  - Process introspection and retrospection
Thank You

Questions?