Using Process Retrospection, Introspection, and Prospection to Support Rework in Software Development (and Other Creative Processes)

Leon J. Osterweil (with Xiang Zhao)
Laboratory for Advanced Software Engineering Research (LASER)
University of Massachusetts Amherst

May 14, 2012
Why define processes?

• Improvement: Largely a static activity
  – Model and Analyze processes to find defects
  – Fix the defects
  – Improve the MODEL

• Guidance: Far more Dynamic in nature
  – Especially useful for executable process definitions
  – Use improved processes to
    • Improve agent coordination
    • Support human performers—especially in creative activities and in REWORK
The Vision

Editing and depictions

Static Analysis

Static Process Improvement
The Vision

Editing and depictions → Static Analysis

Validation Process Model

Executable Process

Yields Better Processes
Fewer Defects
Better Performance
The Vision

Editing and depictions

Static Analysis

Process Execution With Monitoring and Guidance

Validated Process Model

Improvement recommendations

Can Guide and Improve Human Participation and Processes themselves
The Vision

Editing and depictions

Static Analysis

Process Execution With Monitoring and Guidance

- Improvement recommendations

- Operational profiles
  - Historical Data

- Using State and History Make it Better Still

- Input Problem

- Output Results

Validated Process Models
Examples of History-Driven Guidance from Application Domains

• Doctors considering different diagnoses and treatment plans
• Scientists considering different gap-filling models
• Negotiators and Mediators considering different compromises
• Software Engineering...
Helping Humans in Software Development

• Reworking design and coding decisions
  – E.g. Refactoring

• Reworking pre-coding decisions
  – E.g. Spiral Model and ICSM

• Reworking decisions in Formal Verification
  – E.g. Invariants, theorems to be proven
Rework

• Done commonly in software development
  – More than half of all development effort (?)
• Not addressed in most SE textbooks
• Intuitively: “Going back to an earlier phase of software development”
  – Is clearly incorrect
  – Uncovering new requirement during design does not end the design phase (for example)
Rework Is

Repeating activities that had been done previously in an earlier phase, but are now to be in the context of a new phase guided by new contexts, history, and understandings.
Rework Is

Repeating activities that had been done previously in an earlier phase, but are now to be in the context of a new phase guided by new contexts, history, and understandings.
Example Questions During Rework

• What am I doing here?
• How did I get here?
• What did I do the last time(s) I was here?
• How did that work out?
• What alternatives do I have now?
• Which are likely to turn out best?
Example Questions During Rework

- What am I doing here?
- How did I get here?
- What did I do the last time(s) I was here?
- How did that work out?
- What alternatives do I have now?
- Which are likely to turn out best?

Answers rely upon information about current state, past history, future projections
Current Process State

• Similarity to Program State
  – Values of all variables
  – “text pointer” indicates action(s) being executed
  – Concurrency state

• But also
  – Resource and agent Assignments
  – Exception management status

• What else?
Introspection: Examination of Current Process State

• The ability of a process to inspect its current execution state
  – The state “reification”

• And to use the information to influence subsequent execution
Uses of Introspection

• Agent may want to know who else is doing what else?
  – Who should I be coordinating with?
  – Who asked me to do this?

• Excessive iteration count might trigger loop exit

• Deep exception nesting may trigger abort

• Etc.
Uses of Introspection

• Agent may want to know who else is doing what else?
  – Who should I be coordinating with?
  – Who asked me to do this?

• Excessive iteration count might trigger loop exit

• Deep exception nesting may trigger abort

• Etc.

Information managed by Process Interpreter (e.g. Juliette)
Uses of Introspection

• Agent may want to know who else is doing what else?
  – Who should I be coordinating with?
  – Who asked me to do this?

• Excessive iteration count might trigger loop exit

• Deep exception nesting may trigger abort

• Etc.

But current state may not be enough

Information managed by Process Interpreter (e.g. Juliette)
Retrospection

- Ability to support examining past execution HISTORY
- And altering execution based on results
- Some additional components of this state
  - Prior states (values) of artifacts (variables)
  - Trace of execution from start to present
  - Past states of execution stack
  - Etc.
Some Questions this helps answer

• How did I get here?
• What values has this variable (artifact) taken on in the past
• The last time I was here what did I do?
  – And how did that turn out?
• Can I just go back to an earlier decision?

These are central questions about process history that are used to support Rework
Central Focus on ARTIFACTS

• Process definition has three key dimensions
  – Activities
  – Artifacts
  – Resources/Agents

• Most languages focus on Activities

• Essential to balance this with focus on ARTIFACTS
  – In order to deal with hard process issues like
    • REWORK
    • PORTFOLIO MANAGEMENT
Provenance: The Artifact
Examined by Retrospection

- What activities, were done by which agents, when?
- What artifacts were used by which activities to build which other artifacts?
- What iterations have come before, what outcomes?
  - To prevent infinite loops and support human guidance
- How to capture, display, summarize, analyze provenance?
The Data Derivation Graph (DDG)

- The execution trace through a Little-JIL process definition
- Nodes represent process steps
- Edges indicate which artifacts are
  - Derived from executing which step instance
  - Using which artifact instances as inputs
- A history of how the artifacts in a process have been derived
- And more…. What else?
A Factorization Advice Coordination Process

Consider Factorization

Handle Missing Advice

Choose Source

SE Unavailable

Go with only one
Can be executed in a few different ways

• It is often important to know which way it was actually executed
• What agent(s) generated what value(s) and when
• Important to store execution history data persistently
• E.g. we do it with a Data Derivation Graph (DDG)
DDG if both SEs provide advice

Consider Factorization

Factorization Advice

Advice: SE1

advice 1

Advice: SE2

advice 2
If SE₁ Fails to provide advice twice (not shown here), then SE₃ provides advice.

Advice: SE₃

- Advice 3

SE Unavailable

- Advice 3

Consider Factorization

Advice: SE₂

- Advice 2

Factorization Advice
An Example: Using Inspection to support Refactoring

- Refactoring is rework
  - Modification of design
  - Triggers modification of code
- Should be guided by past history
  - Why refactor?
  - Results of previous refactoring
- Help from state inspection is useful too
  - What changes have already been made?
  - Who else is doing what else?
The Refactoring Project (Zhao)

- Define a Refactoring Pattern in Little-JIL
  - Leads to deeper understanding
- Execute the Process using
  - Juliette (Introspection)
  - DDGs (Retrospection)
- Becomes a refactoring support tool
  - Leads to understanding of additional needs
- Define more refactoring patterns
  - More needs
The “Separate Query from Modifier Refactoring” Pattern
Where Some Rework Originates
Where Some Rework Originates

Elaborate on this
Where some of the Rework is
Where some of the Rework is
More complicated Rework
More complicated Rework

Arbitrarily deep recursion here

- Retrieve Reference files
- Update Reference
- Retrieve source file
- Replace original call to call the query
- Add a call to original method before the query
- Update References
- Handle Reference Compilation Error
- Handle Reference Unit Test Failure
- Compile
- Run unit tests
- Update Reference
- Handle Reference Compilation Error
- Compile
- Run unit tests
Recursion-Driven Rework

- Iterative correction of a mistake is rework
- Well-represented by recursion
  - Each iteration occurs in the context of all the previous ones
  - Understandings gained need to be made available
  - The context needs to be a parameter to such steps
- Using the DDG to represent context
  - Access to DDG passed to the recursive step
The DDG Project (Work being led by Barbara Lerner and Xiang Zhao)

- Defined templates for translating Little-JIL step executions into DAG fragments
- Gluing software for building DDGs from them
- Incorporates scoping, nesting, hierarchy information
- Links to previous values of artifacts
- Detailed history is inferrable
- Can generate DDGs dynamically while process is executing
Example

DDG

Little-JIL

Legend
Actual Generated DDG (From Ecology Process Definition)
Enlargement of a Piece
Refactoring Process DDG Details
An Example Method to be Refactored

```
Listing 1: CheckingAccount.java
1  public class CheckingAccount{
2   private float balance;
3   public CheckingAccount(float balance) {
4     this.balance = balance;
5   }
6   public float checkForBalanceAndWithdraw(
7       float amount) {
8     if (balance >= amount) {
9       balance -= amount;
10      System.out.println("Widthdraw done.
11         Current balance: " + balance);
12      return balance;
13     } else {
14       System.out.println("There is not
15         enough balance!");
16      return balance;
17     }
18   }
19  }
20  public void deposit(float amount) {
21    balance += amount;
22  }
```
What the User Sees
Some More Screen shots
Is this how to present this to humans?

- What artifacts to show?
- How to arrange them for readability?
- How to show time sequencing (e.g. of variable values?)
Approach

• Output DDG information as relations

• Store in a database
  – Using one developed by Margo Seltzer [Harvard U]
    • A DB of triples representing edges
  – Specialized for scientific datasets (a lot of repetition)

• Use Seltzer’s query language to retrieve

• Automated layout program to display
More complicated Rework

These can be multiply instantiated

How to handle this ripple effect?
Ripple Effects

• More complex refactoring creates ripple effect
  – One change may have many consequences
  – Changing a consequence has its own consequences
  – Consequences may interfere with each other

• How to decide which consequence to pursue?

• How to predict which changes are most effective?
The importance of Prospection

- Helps to know where things are going in the future
- Have used this in resource scheduling (Xiao)
  - Prospection to project future resource needs
  - Optimize resource utilization
  - Project Portfolio Management (PPM)
- Also useful in managing ripple effect
  - Which choices have which consequences
  - Which changes address multiple modification needs?
  - What changes create more ripples?
What makes a language Prospective?

- Ability to support examining possible FUTURE execution states
- And altering execution based on results
  - E.g. providing guidance on choices
- Some components of FUTURE state
  - Possible future execution paths
  - Possible states (values) of artifacts (variables)
  - Expected concurrency states
    - Including races, deadlocks, etc.
  - Projected needs for resources
  - Etc.
Prospection used in TWINS Incremental Resource Scheduling (Joint UMass/ISCAS with Junchao Xiao)

• An approach that exploits detailed specifications of process activities, their needs for resources, and the characteristics of the resources themselves

• Decomposes the overall resource scheduling problem into a series of dynamic reschedulings happening at selected times and covering sets of activities
  – Each driven by prospective analysis of future process execution
Scheduling activity set constructor builds Prospective Execution View

- Assemble all of the information needed in order to make scheduling decisions
- First generate Resource Utilization Flow Graph (RUFG) from Little-JIL process
Incremental Resource Scheduling Framework

- Time Line
  - New Requirement
  - Rush Order
  - Exceptions
  - Resources Change
  - Scheduling Window

- Rescheduling Indicator
  - Scheduling Activity Set Constructor
    - Process Asset
    - Resource Repository

- Scheduler
  - Un-executed Activities
    - Activities Needed to be Scheduled
  - Scheduling Parameters
  - Scheduling Result

- Current (introspective) Execution State

Process Execution System
Incremental Resource Scheduling Framework

```
<table>
<thead>
<tr>
<th>Process Asset</th>
<th>Resource Repository</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

- **Computed (prospective) Process State**
- **Current (introspective) Execution State**

Diagram:
- **New Requirement**
- **Rush Order**
- **Exceptions**
- **Resources Change**
- **Scheduling Window**

- **Time Line**
- **Rescheduling Indicator**
- **Scheduling Activity Set Constructor**
- **Scheduler**
- **Execution State Feedback**
- **Un-executed Activities**
- **Activities Needed to be Scheduled**
- **Scheduling Result**
Need New Focus on Prospective DDG
Scheduling activity set constructor builds Prospective Execution View

- Assemble all of the information needed in order to make scheduling decisions
- First generate Resource Utilization Flow Graph (RUFG) from Little-JIL process
Many more uses for Prospection

• Agents want to know the consequences of their actions
• Where is process headed
• What are likely outcomes
• Etc.
Some Problems Similar to Retrospection Problems

• Voluminous data
• How to create it
• How to organize it
• How to store it
• How to display it
Rework In A Waterfall Lifecycle
Rework In A Waterfall Lifecycle
Use of History/Retrospection In A Waterfall Lifecycle
Rework And History are Clearly Implied in Spiral Iterations Too
Rework And History are Clearly Implied in Spiral Iterations Too
Summary

• Rework is a key part of all creative activities
  – Supporting it with process execution is very useful

• State reification, history generation, prospective execution are all very important supports for rework

• Process languages need to support them with introspection, retrospection, prospection

• Raise hard and important problems
Research Agenda Items

- How to modify process languages to support all of this
- How to support run-time communication and synchronization between model and real-world processes
- How to store, manage, and display state information (present, past, future)?

We are working on these, but need help with them