Tracking dependencies between code changes

Lucas Godoy

Scenario

- It’s common for software developers to work in more than one problem at the same time
  - For example, to fix a critical bug in the middle of a big refactor
  - Good practices recommend to perform atomic commits

Define atomic

- Ideally, an atomic commit should only affect a single aspect of the system
- This improves understandability, makes easier bug identification and requires less effort to rollback
- But the atomic commit policy is not enforced by tools

The problem

- The developer has to cherry-pick changes manually (by using git-add -i)
- But with traditional VCS, changes are scattered in a large amount of text
- And detailed information about the sequence of changes is lost

Objective

- To aid the developer in the cherry-picking process
- The resulting change-set should be atomic
- Limitations: not fully automatic, not 100% accurate in all cases

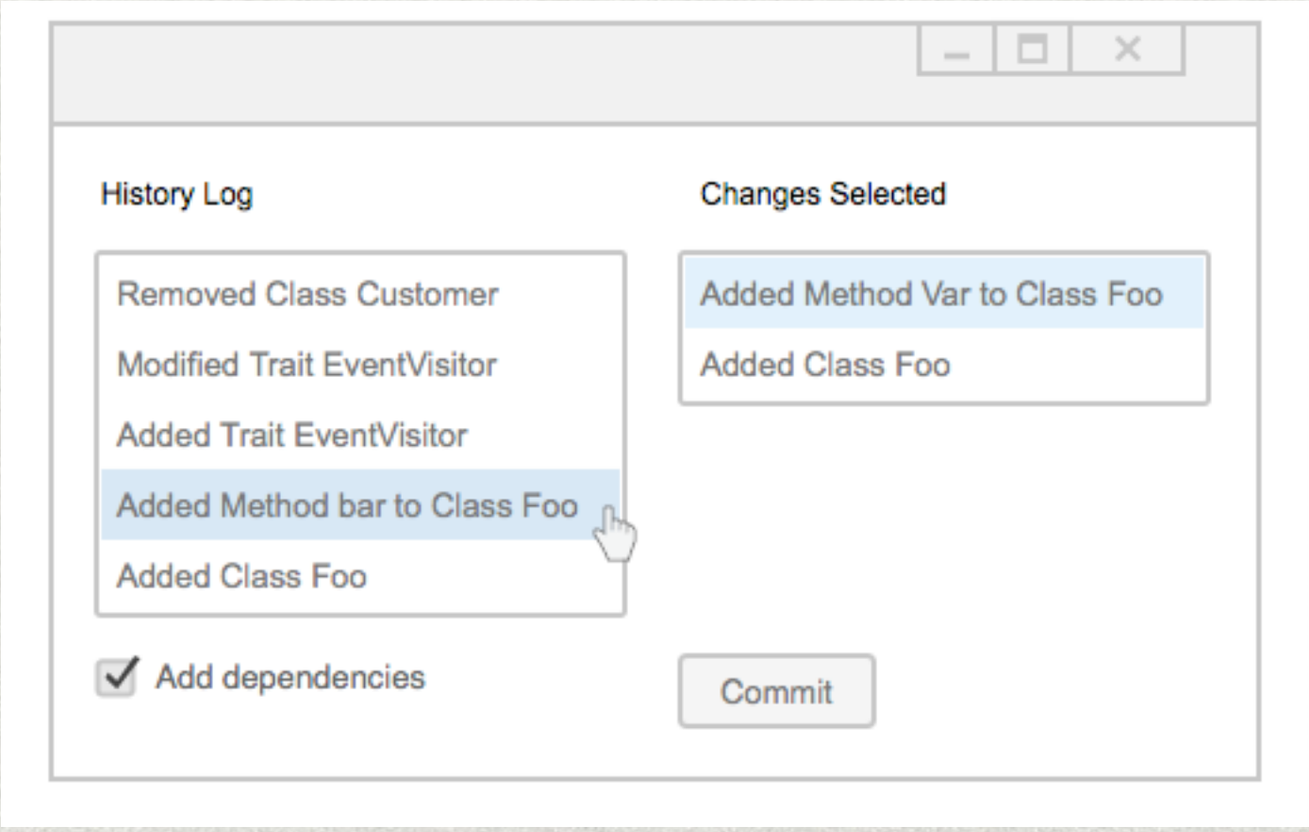
What we would like

- Changes modeled as objects and recorded as they happen
- To track dependencies between those objects to define the atomic change set we're going to commit
- And dependencies should also be modeled as objects

What tools do we have?

- ChangeSorter: log of executable statements, not all entities are objects, coarse granularity
- CoExist: coexisting versions, continuous testing, change-oriented, no dependency tracking
- Epicea: change-oriented, no dependency tracking
- JET: snapshot-based, tracks dependencies

Proposed solution



- Do dependency analysis based on the system history

Structural dependencies

- The suggested rearrange of changes is based on structural relationships between the entities
- 3 types of dependencies
- Class hierarchy dependencies
- References to variables and classes
- Message sends

Examples

```
Object subclass: #AbstractTimeZone
   instanceVariableNames: ''
   classVariableNames: ''
   poolDictionaries: 'ChronologyConstants'
   category: 'Kernel-Chronology'


Trait named: #TClass
        uses: TBehaviorCategorization
        category: 'Traits-Kernel-Traits'

AbstractTimeZone >> printOn: aStream
        super printOn: aStream.
        aStream
           nextPut: $(;
           nextPutAll: self abbreviation;
      nextPut: $).
```

Message sends

- 3 kinds with different scope for a candidate set
- Messages sent to self are restricted to the current class hierarchy
- Messages sent to super are restricted to the upper part of class hierarchy
- Messages sent to classes are restricted to the class side of the given class

Unknown sends

- If the message sent don't fall into any of the previous categories, we have to put all implementors in the candidate set
- This is what we call an unknown send
- Basically any polymorphic message can lead to false positives in the candidate set

Implementation

- One visitor for Epicea events
- Another visitor for AST nodes to find dependencies in methods
- Early prototype called Tracks: http://smalltalkhub.com/#!/~LucasGodoy/Tracks

Future work

- Dealing with shadowing
- History reconstruction
- Integration with Epicea
- Dependency transitivity
- Visualization of dependencies (Telescope?)
- Performance test & optimization

In summary

- To make an atomic commit by cherry-picking changes can be time consuming
- We propose a tool to make it easier
- The tool suggests additions to the change-set by finding dependencies between changes made to the system
- You can send feedback and comments to godoy.lucas@gmail.com