valeria murgia
product design lead - caesar systems

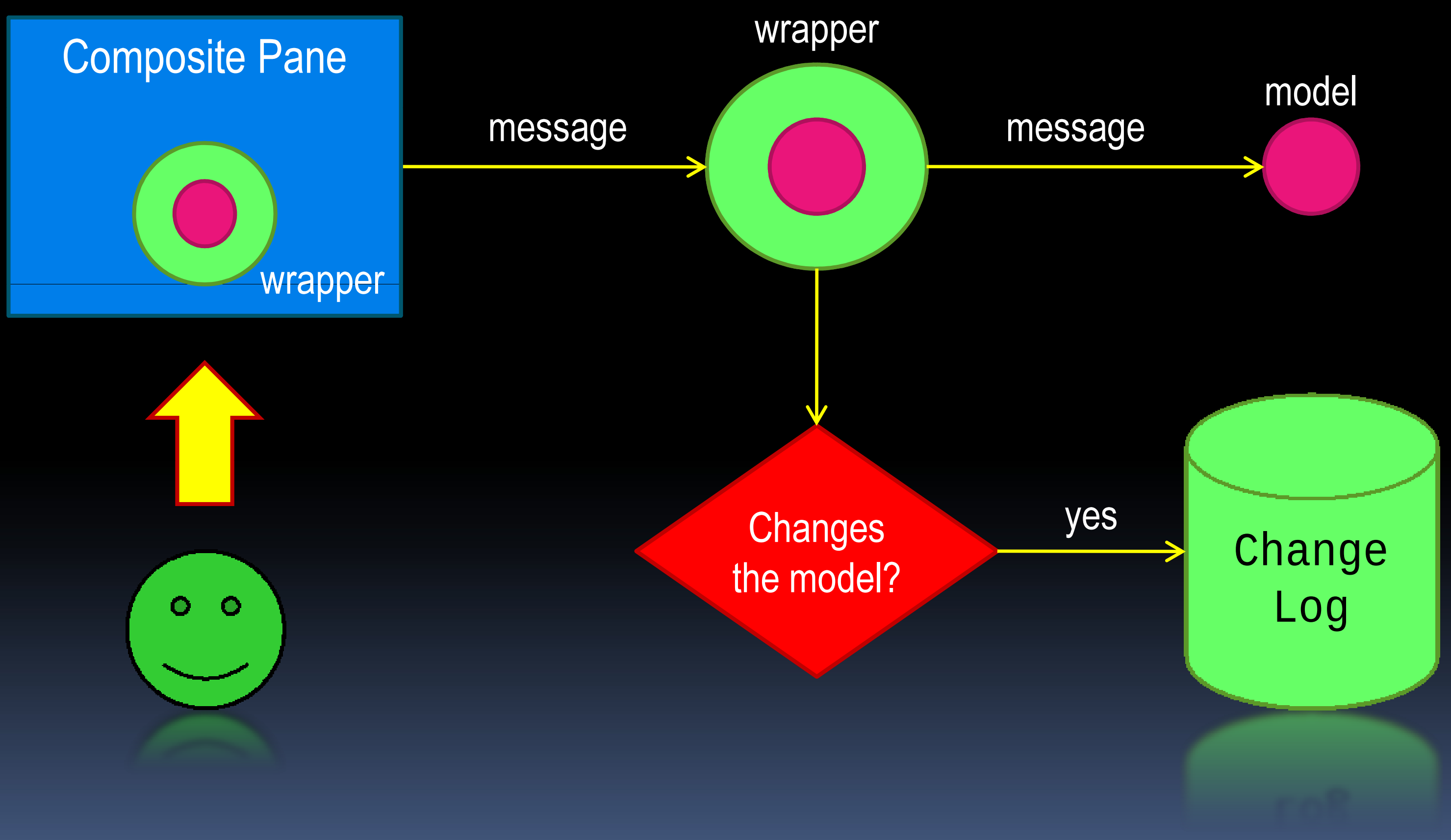
leandro caniglia
director of development - caesar systems

MAKING THE MOST OF USER CHANGES

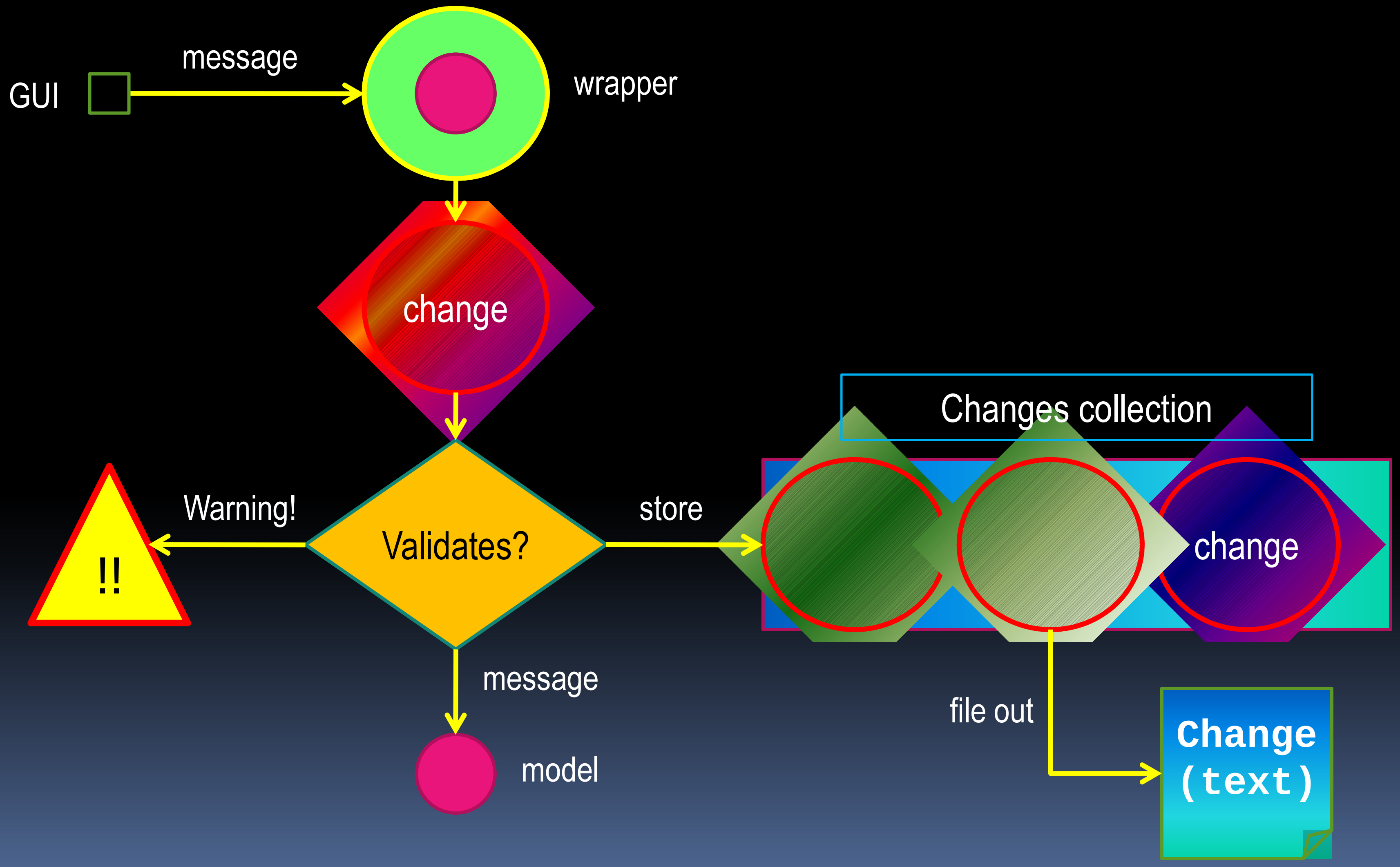
Demo

- Use the software
- Generate some changes
- Show the changes browser
- Replay them

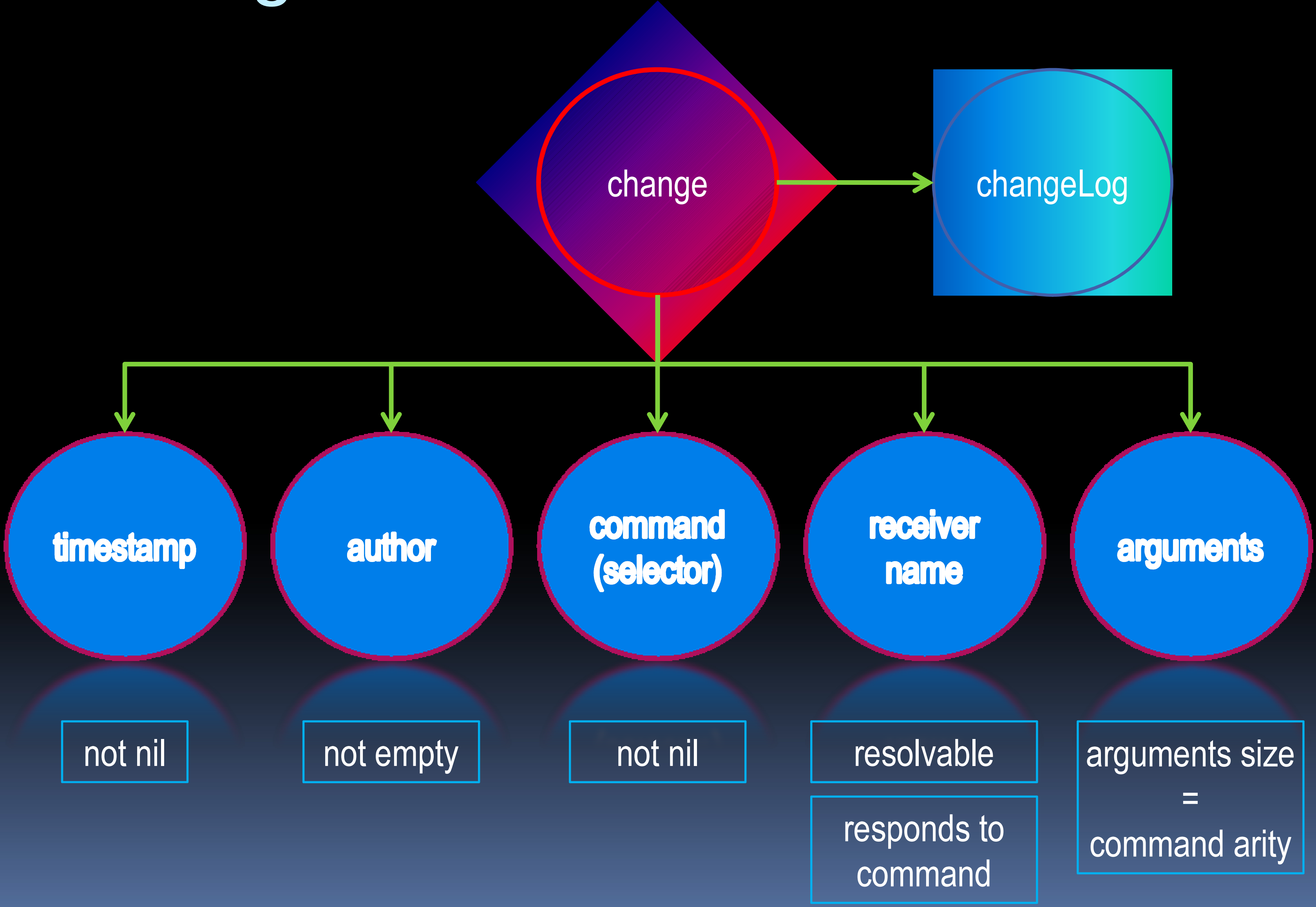
Logging user changes



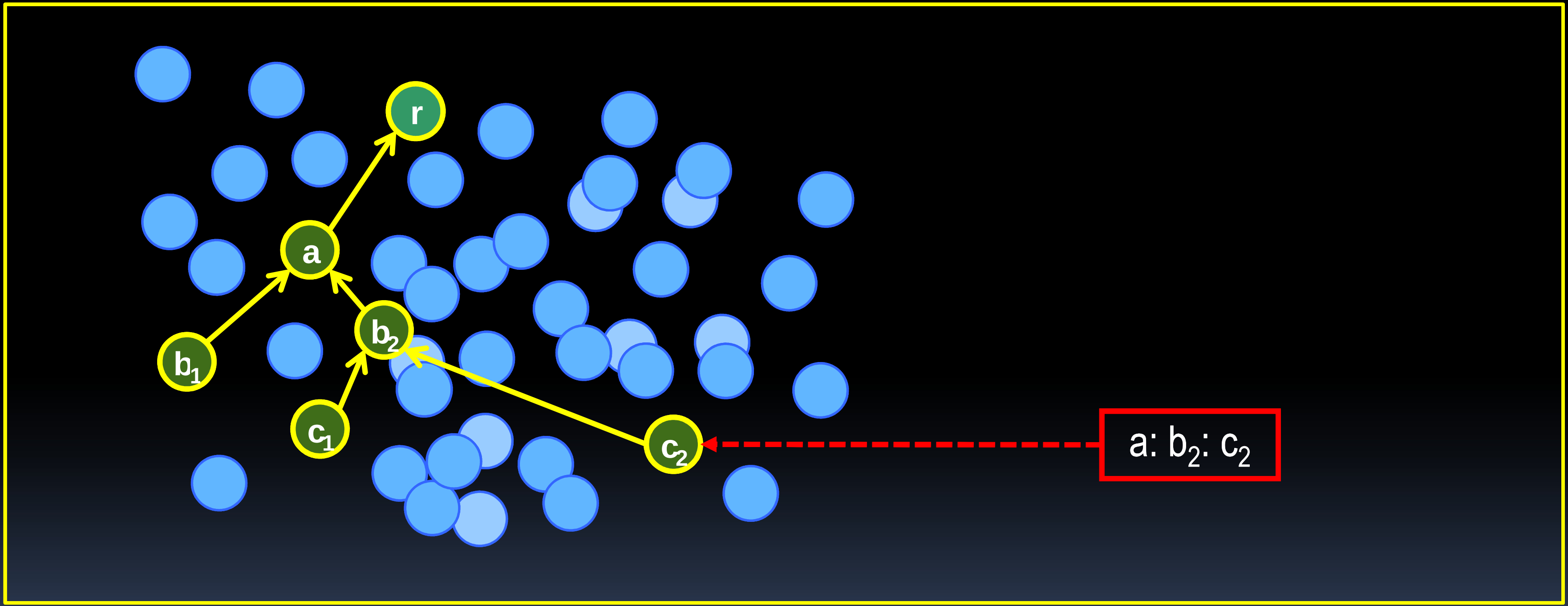
Command logging



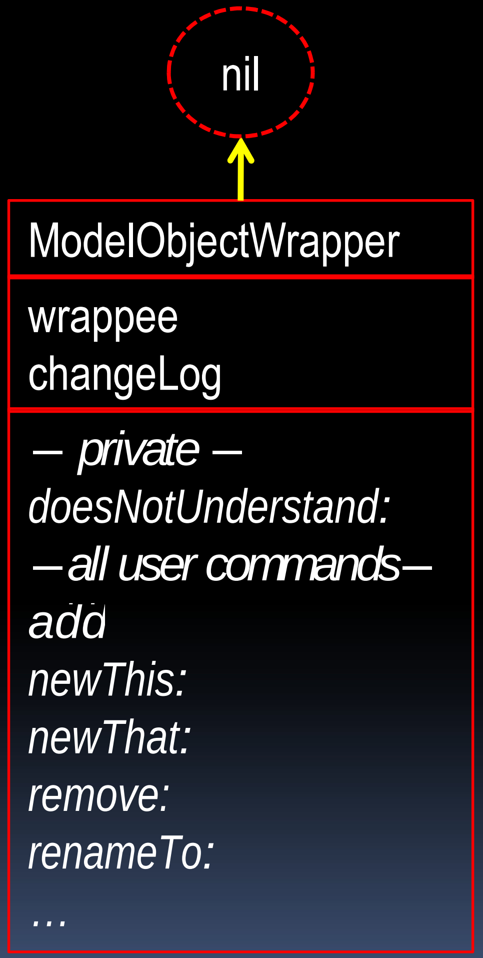
Change structure



Naming objects



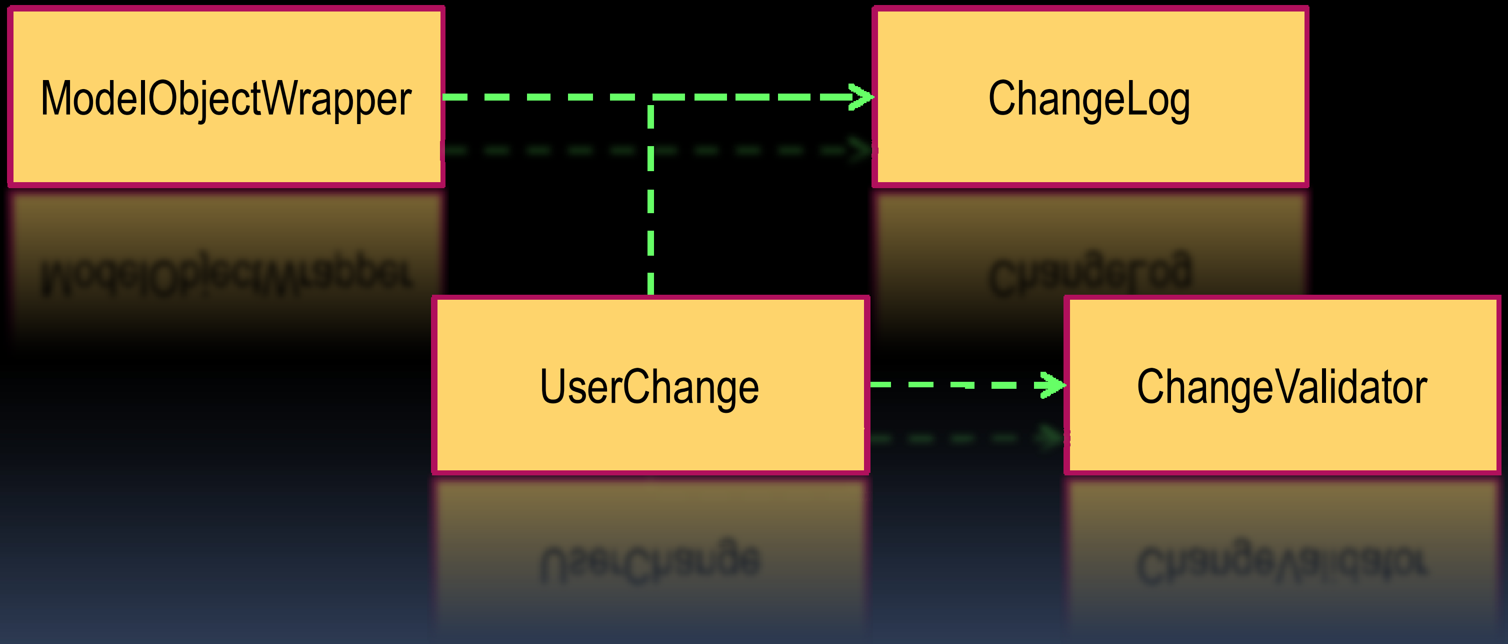
Creating changes



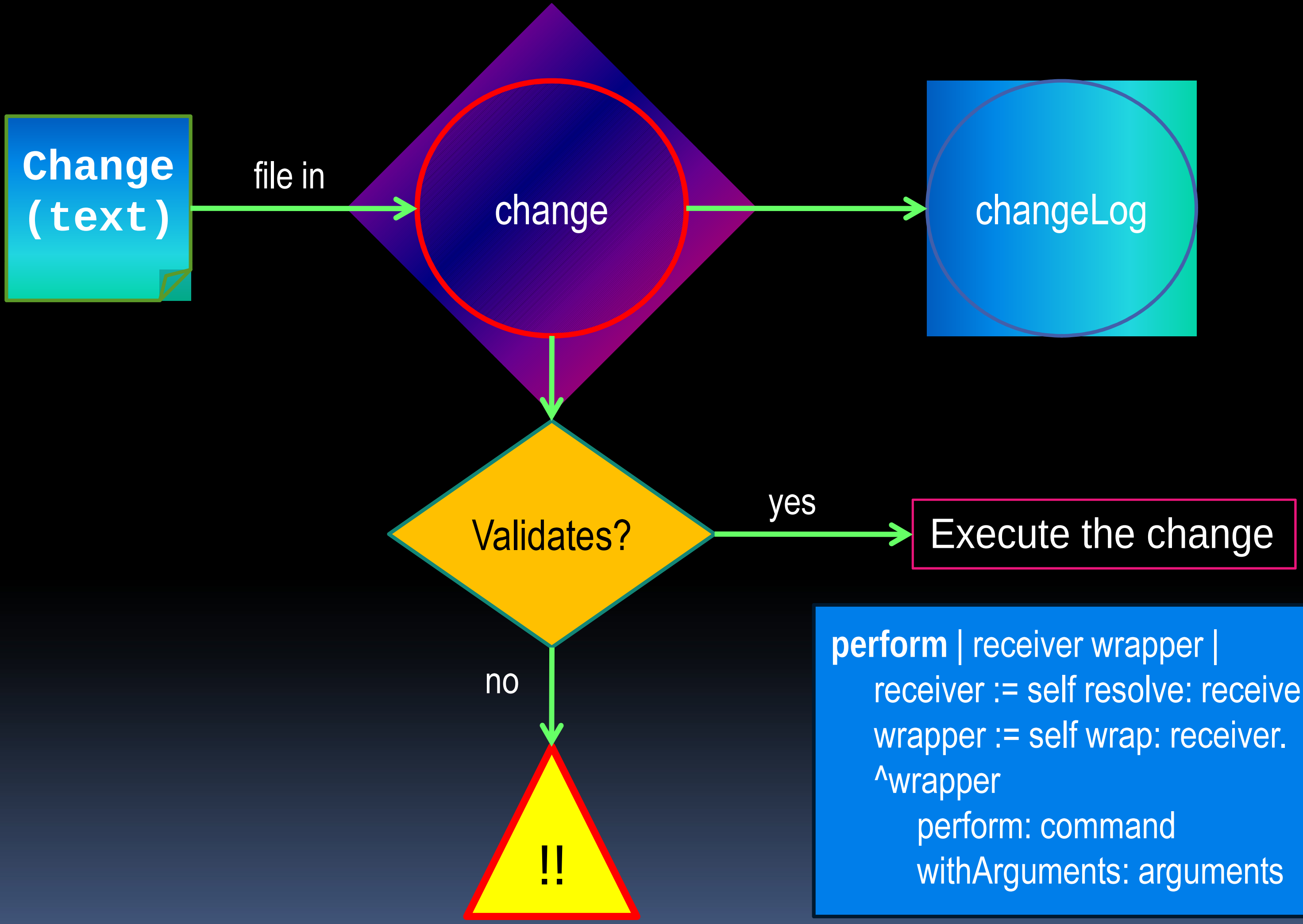
```
doesNotUnderstand: aMessage
    | selector |
    selector  := aMessage selector.
    (self shouldBuildMethodFor:  selector)
        ifTrue: [self buildMethodFor: selector]
        ifFalse: [aMessage receiver: wrappee].
    ^aMessage perform
```

```
renameTo:  arg
    changeLog newChange
        wrappee: wrappee
        command: #renameTo:
        argument: arg.
    ^wrappee renameTo: arg
```

Classes involved



Replaying changes



```
perform | receiver wrapper |
    receiver := self resolve: receiverName.
    wrapper := self wrap: receiver.
    ^wrapper
        perform: command
        withArguments: arguments
```

Applications

- Recovery log
  - all the time save every change on disk
- Auditing
  - who changed what, when and how
- Local redo (can be used for undo)
  - right click on any object and list all its changes
- Scripting
  - use the changes system as a scripting language

Applications continued

- Demos & Tutorials
  - demo your system by replaying changes
- Overcome back compatibility issues
  - recreate old projects from their changes
- Merging
  - merging changes is easier than merging objects
- User support
  - solve the user's problem and send back the changes

Applications continued

- Bug reporting
  - send the changes that exhibit a defect
  - don’t know what you did? look at the changes!
- Testing
  - look at the changes to write unit tests
- Regression
  - build a library of scripts to test your system
- Learning (new programmers)
  - use the changes as debugging entry points

Applications continued

- Metrics
  - count the number of commands your users can perform
  - how many keystrokes does your software require?
  - measure user dedication and productivity
  - which areas of your software are more heavily used?
  - understand users' workflows
  - discover bad practice patterns

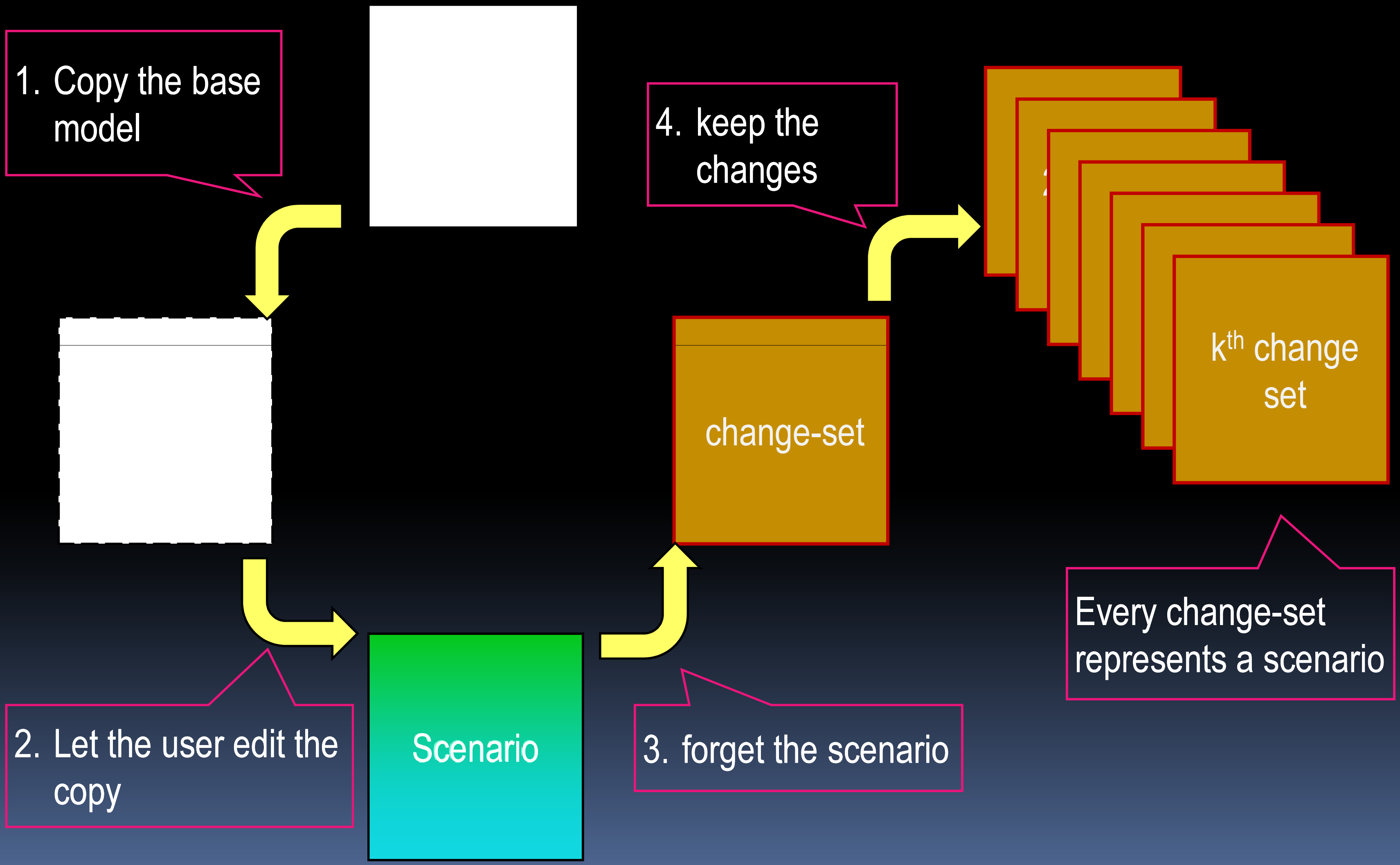
Applications continued

- Teamwork
  - combine changes from different contributors into the same model
- Database conflicts
  - let users recover conflicting changes that did not get committed
- Database: automated check-in/check-out
  1. download a project from the database
  2. work at home
  3. apply your changes back to the repository

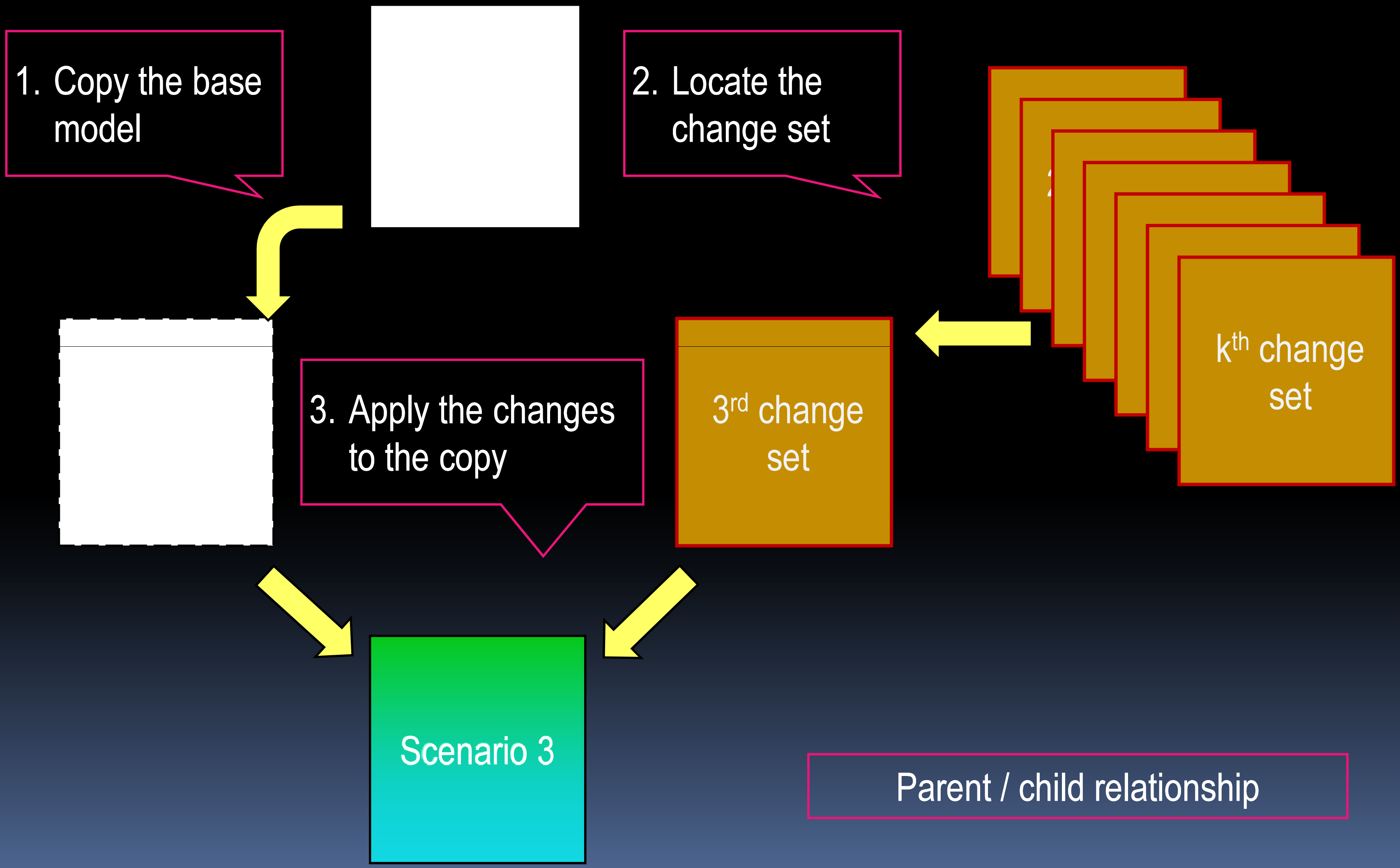
Demo

- Open a project
- Make some few changes
- Create two scenarios
- Edit & change the scenarios
- Show the changes

Decision analysis: scenarios



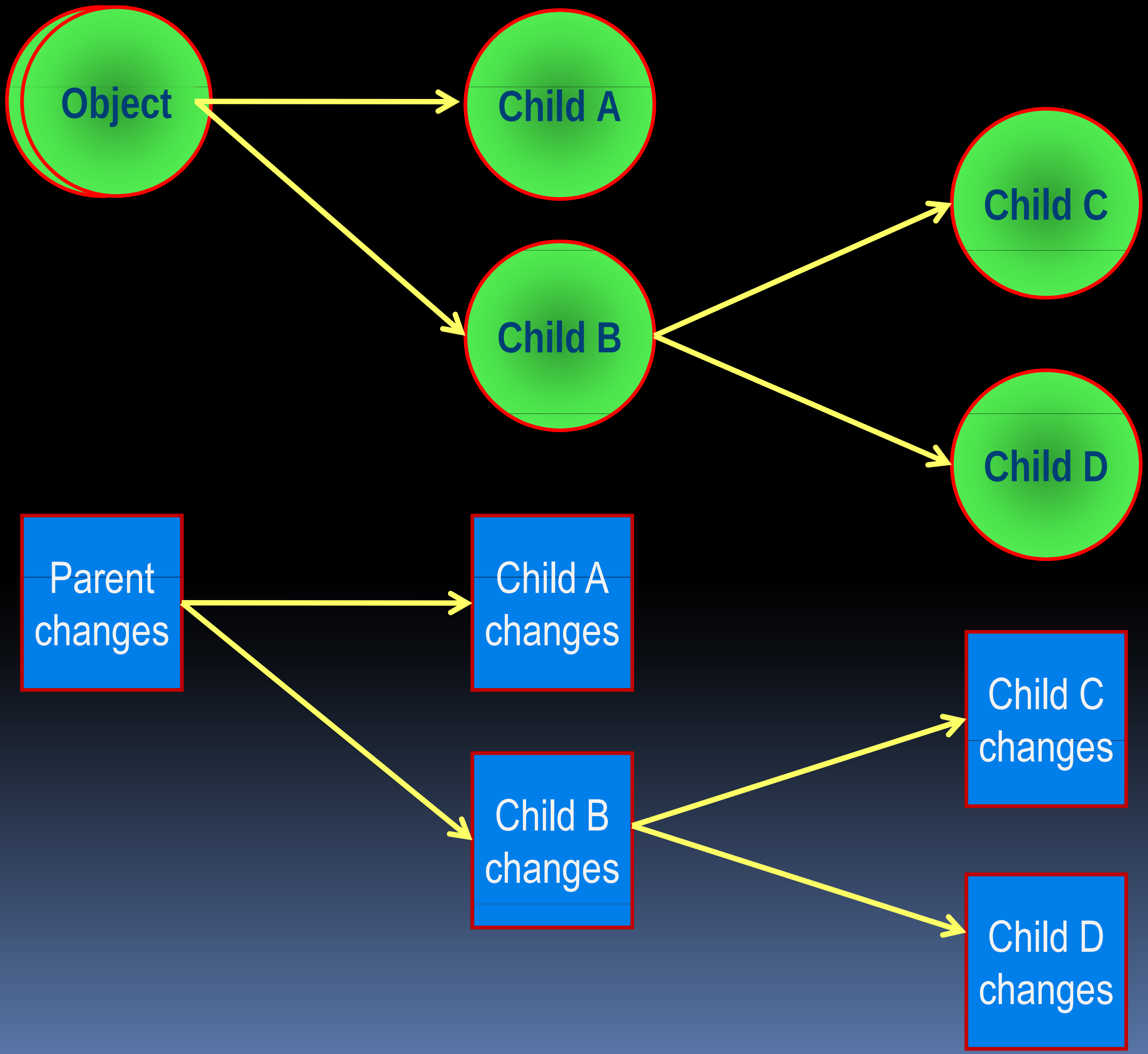
Scenarios continued



Demo

- Decision Tree demo

Decision trees



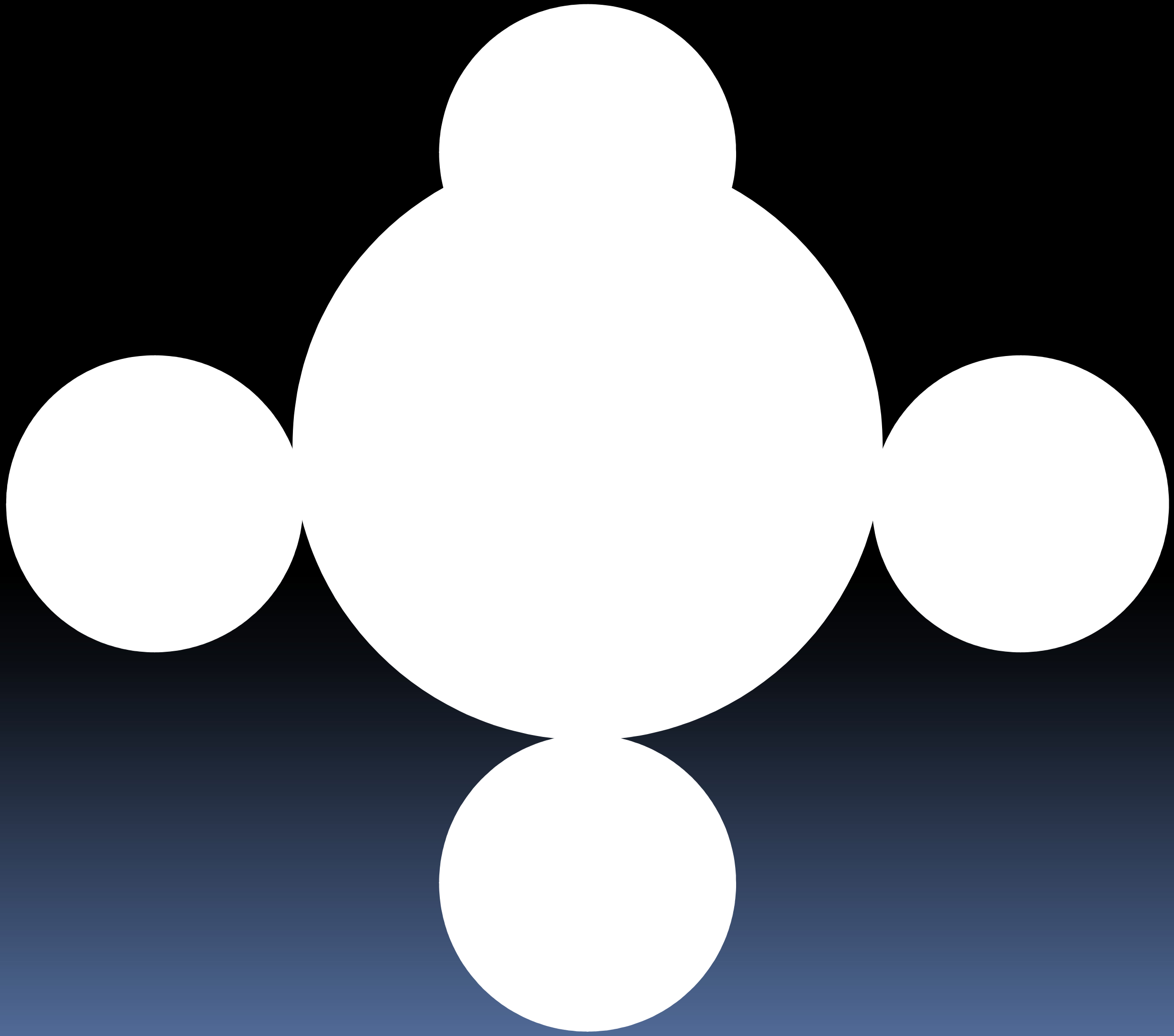
Applications continued

- Scenarios
  - associate change sets with scenarios for what-if analysis
- Decision trees
  - Organize change sets under a hierarchical structure
- Monte Carlo simulations
  - create one scenario for every random sample

Demo

- Monte Carlo demo

Influences



Questions

- Have you implemented all the applications described here?
- Have you used the changes system to analyze the workflow of end-users?
- Does the changes system impact the performance?
- What's the overhead for programmers?
- What if arguments are not literals?
- Can your system log any user action?